INTELLIGENT MACHINES AS HIERARCHICAL STOCHASTIC AUTOMATA

By

Pedro M. U. A. Lima

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Electrical, Computer and Systems Engineering

Rensselaer Polytechnic Institute
Troy, New York

August 1994

CONTENTS

LIST OF TABLES

LIST OF FIGURES

À memória da minha mãe

e ao futuro do meu filho

ACKNOWLEDGMENTS

I would like to thank George Saridis for all his support and encouragement, and for the many exciting discussions we had. I am looking forward to our future cooperation. I would also like to express my gratitude to all members of my doctoral committee for their helpful suggestions.

An overseas thanks to all my friends and colleagues of Instituto Superior Técnico in Lisboa, especially to José Alberto Victor for his friendship and many long-distance brainstorms, Paulo Oliveira for taking care of my paperwork in Portugal and his friendship, Carlos Pinto Ferreira for justifying the saying “better later than never”, and João Sentieiro, whose encouragement made all this possible. I also owe a word of acknowledgment to three partners in this american adventure: Rogério Rodrigues, Carlos Bispo and João Costeira, for all their support. Many thanks also to João Tasso for reviewing several drafts of this work and for a new friendship. A special thanks to Internet, without which many of these thanks would not be possible.

Thanks also to my friends and colleagues in the 8th floor of CII, especially the members of the “Intelligent Machines Gang” (Joe Musto, Randy Beard and Linden Carmichael) for many insightful discussions, as well as to Deepak Sood, for teaching me many secrets of the UNIX system and CIRSSE Testbed.

The computer code for the simulation of the PUMA dynamics and the graphical display of Case Study 2 setup was written at CIRSSE by S. Murphy, D. Swift and Keith Nicewarner.

Last, but not the least, a huge thanks to all my family for their never ending support and patience. Part of this thesis is theirs too.

ABSTRACT

Most of the work done in the last few years by several researchers towards building Autonomous Intelligent Controllers quite often mentions the need for a methodology of design and a measure of how successful the final result is.

A new design methodology is introduced in this thesis for improvement of performance of Intelligent Controllers developed by the *Analytic Theory of Intelligent Machines* proposed by Saridis. The translation interfaces of a 3-level *Hierarchical Goal-Directed Intelligent Machine* (HGDIM) are modeled by a 2-stage *Hierarchical Learning Stochastic Automaton* (HLSA). The HLSA is an original extension of the Generalized Learning Stochastic Automaton (LSA) proposed by K. S. Fu and his associates. The decision probabilities at the two stages are recursively updated from the *success* and *failure* signals received by the bottom stage whenever a primitive algorithm of the HGDIM is applied to the environment where the machine operates. Under this learning scheme, the probability of selecting the optimal tasks and primitive algorithms is proven to converge to 1 with probability 1. An *optimal action* (task or primitive algorithm) is defined as the action which *minimizes* a *cost function* recursively updated through feedback. This cost function of an action has two terms: one is the *cost* of applying the action, and the other is the complement of the *reliability* of the action.

Other novel contributions of this work include a coherent analytical measure of algorithm cost and reliability, a new measure of performance for HGDIMs, and an original Hierarchical Reinforcement Learning Scheme for HLSAs, based on the bottom-up propagation of the cost function.

Results of simulations show the application of the methodologies to the Operations Management of a Glass Furnace, and Intelligent Robotic Systems.

CHAPTER 1
Introduction

The evolution of Control theory and applications in the past 60 years points towards increasingly complex systems. Conventional control design techniques assume the existence of mathematical models that capture the dynamical behavior of the systems to be controlled. The evolution from input-output to state-space models and from fixed feedback controllers to adaptive and robust controllers, capable of dealing with temporal changes and uncertainty about system parameters, corresponds to an increasing complexity of the control design process, motivated by an increasing complexity of design specifications and system models. As the mathematical models become more involved, so does the number of simplifying assumptions, to keep the model tractable. This may lead to control design strategies appropriate to the models, but inappropriate to the real systems they model, since one hardly corresponds to the other.

Since the late 60s, various strategies were proposed to address the control of complex systems. K. S. Fu[17] was probably the first to write about Learning Control Systems and to coin as Intelligent Control Systems those systems of interdisciplinary nature, in the intersection of Artificial Intelligence and Automatic Control. An increasing number of other researchers have developed applications and theory in the new discipline by introducing new ideas such as neural control[3, 25, 57], fuzzy control[94, 28, 35], hybrid control[77] or hierarchical control[3, 88]. Intelligent Control techniques particularly qualify for applications to Robotics and Process Control, due to the need to coordinate a diverse and large number of sensors and actuators. They differ from "conventional" techniques by aiming to attain higher degrees of *autonomy*, thus dealing with higher *uncertainty*. To accomplish this, Intelligent

Controllers need to adapt to unexpected situations, and learn how to solve problems associated in the past only with human decisions. In doing so, they attempt to emulate mental faculties which are believed to be important attributes of human intelligence. The name *Intelligent Control* comes from this attempt.

This thesis introduces a design methodology for Intelligent Controllers based on the *Analytic Theory of Intelligent Machines* developed by Saridis[69]. The methodology relies on the existing knowledge about designing the different subsystems composing an Intelligent Machine. Its goal is to provide a measure of performance applicable to any of the subsystems, and use that measure to learn on-line the best among the set of pre-designed alternatives, given a state of the environment where the machine operates. Different designs can be compared using this novel approach.

1.1 Motivation

Most of the work done in the last few years towards building Intelligent Controllers[77, 72, 3, 53, 10] quite often mentions the need for a methodology to design an Intelligent Machine (IM) and a measure of how successful the final result is. An analytic design based on measures of performance recursively improved through feedback assures some degree of certainty about the measurability, repeatability and verifiability of that design. This point of view also emphasizes an approach based on Control Systems Theory, even though the whole design is based on other multidisciplinary contributions, such as Artificial Intelligence and Operations Research.

The architecture for an IM proposed by Saridis is based on a 3-level hierarchy, where more abstract actions are taken at the top **Organization Level** and more precise actions are taken at the bottom **Execution Level**. Given an *external command* (goal), the Organization Level is responsible for sequencing the pre-defined *events* into a *task*. The Execution Level executes a detailed translation of the task, generated by the intermediate **Coordination Level**. This level further decomposes the events composing the task, and distributes them by a number of coordinators specialized in specific sub-tasks, such as vision, motion or path planning in an Intelligent Robotic System. The coordinators invoke the Execution Level *primitive algorithms* to precisely execute the task.

When dealing with very large systems, there is always uncertainty about the behavior of the system to be controlled. Hence there is always uncertainty about the result of a given command sent to the controlled system. *Uncertainty* is experienced by all levels of the Intelligent Machine:

- at the **execution level**, there is uncertainty in terms of overshoots, position and velocity errors, confidence interval of an object pose estimated by a computer vision algorithm, degree of accuracy to which a planned path or trajectory follows a given set of knot points, and other similar features, since mathematical models never match exactly the actual controlled system. Implemented by *primitive algorithms* (e.g. controllers, image processing), feedback reduces the uncertainty about the above mentioned features thus reducing the entropy of the system, as suggested by Zames[95].

- at the **coordination level**, there is uncertainty in terms of the success of each of the *primitive events* composing a *task*. In an Intelligent Robotic System, primitive events typical of this level are grasp strut, plan path, move manipulator, locate object. A primitive event represents a problem whose solution is implemented by a primitive algorithm.

- at the **organization level**, there is uncertainty in terms of the success of the *task* planned.

Thus, an Intelligent Machine should use the environment feedback to reduce along time the uncertainty about the success of its actions and the uncertainty about which tasks and primitive algorithms to select. Globally, performance is related to uncertainty. Reducing the uncertainty improves the performance. However, the cost of reducing uncertainty must also be taken into account.

The boundary between the *environment* where an IM operates and the IM, is frequently application-dependent. However, the general rule is that the IM includes all algorithms necessary to cope with some goal. The environment is the system controlled by the machine, including algorithms which inspect the environment state and the success of actions of the machine, and hardware which physically belongs to the wide-sense IM. As an example, consider an Intelligent Mobile Robot moving inside a room. The room and its objects, and the motors, wheels, cameras and sensors of the Mobile Robot are part of the environment. The planning, learning, decision making and execution algorithms (e. g. motion controllers, vision and sensor fusion algorithms, path planners) are part of the IM.

This thesis proposes a methodology for performance improvement of *Hierarchical Goal-Directed Intelligent Machines* (HGDIMs) based on *Hierarchical Reinforcement Learning*. The translation interfaces of the 3-level architecture proposed by Saridis and Valavanis[88] are modeled by a 2-stage *Hierarchical Learning Stochastic Automaton* (HLSA), as sketched in Figure 1.1. The HLSA includes a *Hierarchical Reinforcement Learning Scheme* which recursively updates the decision probabilities at the two stages from *success* and *failure* signals received by the bottom level whenever an action of the HGDIM is applied to the environment where the machine operates. Under this learning scheme, the probability of selecting the best tasks and primitive algorithms is proven to converge to 1 with probability 1 (w.p.1).

This work is an extension of the framework of the *Analytic Theory of Intelligent Machines* developed by Saridis *et al*[71, 68, 54, 88, 93, 50, 37]. The main contribution of this thesis to the *Analytic Theory of Intelligent Machines* is the use

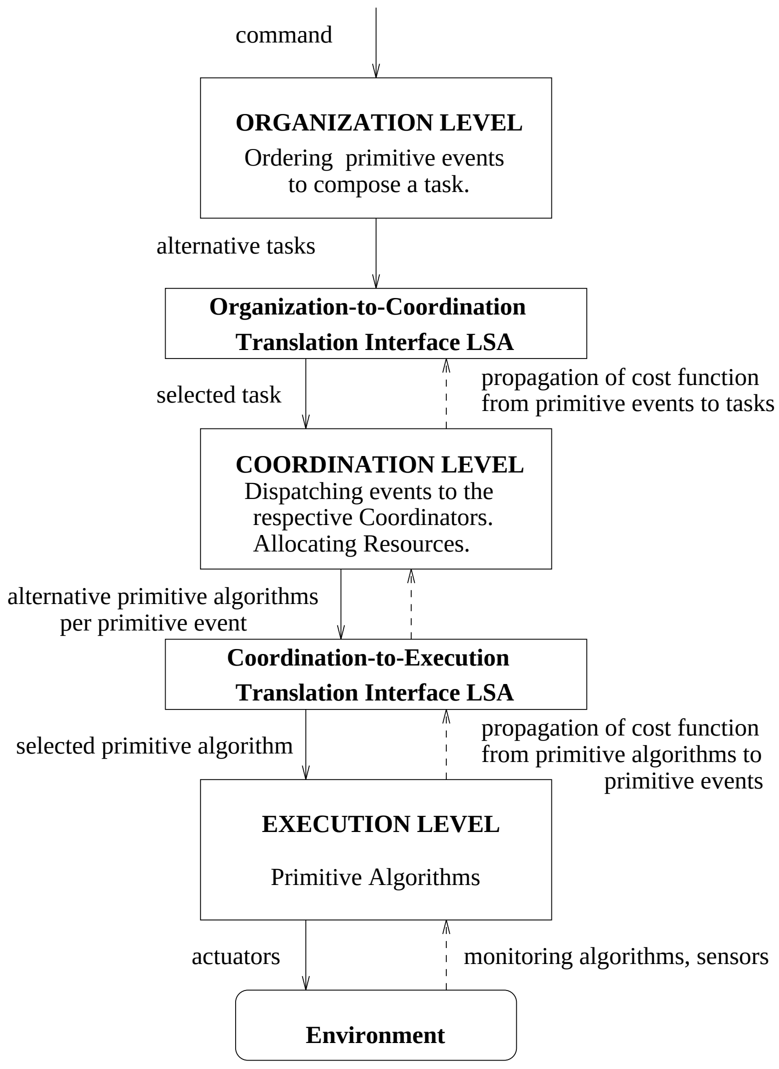
command
ORGANIZATION LEVEL
Ordering primitive events to compose a task.
alternative tasks
Organization-to-Coordination Translation Interface LSA
selected task
propagation of cost function from primitive events to tasks
COORDINATION LEVEL
Dispatching events to the respective Coordinators.
Allocating Resources.
alternative primitive algorithms per primitive event
Coordination-to-Execution Translation Interface LSA
selected primitive algorithm
propagation of cost function from primitive algorithms to primitive events
EXECUTION LEVEL
Primitive Algorithms
actuators
monitoring algorithms, sensors
Environment

Figure 1.1: Hierarchical Learning Stochastic Automaton and Hierarchical Goal-Directed Intelligent Machine.

of feedback from the environment to update a specific cost function which evaluates the performance of all three levels of the hierarchy. This evaluation guides the update of the decision making structure. The cost function has two terms: computational cost and reliability, which are defined coherently.

Other novel contributions of this work include an original hierarchical extension of Fu's Generalized LSA, the corresponding Hierarchical Reinforcement Learning Scheme and the original modeling of a HGDIM by a HLSA.

Previous results, referenced in chapter 2, established a general architecture for Saridis' IM and detailed this architecture for the different levels. However, the flow of feedback through the hierarchy with the purpose of improving the overall performance by updating the decision making structure, has never been detailed for the complete hierarchy. Furthermore, even though the general goal is to decrease *entropy* at all levels, and reliability has been proposed as an equivalent measure of entropy[50], neither has computational cost ever been included in the cost function, nor has a recursive estimate of reliability been considered.

To include those features, a coherent measure of *computational cost* and *reliability* of an algorithm will be introduced before the deployment of the learning and decision making methodology. The two measures are later combined in a *cost function* which is estimated as part of the learning algorithm. However, it can be used independently for off-line design too, extending the methodology proposed by McInroy and Saridis[49], and Musto and Saridis[55].

This approach has the advantage of providing a guideline for the solution of several different problems, since it is based on measures of *reliability* of an algorithm (defined as the probability that the algorithm will meet some set of specifications in a given state of the environment) and *computational cost* of an algorithm (i.e. the number of resources used by the algorithm to solve a problem with the required reliability). Computational cost includes general measures such as computation time, memory used, number of processors used or mean square error with respect to some desired set point. These are sufficiently general measures in the sense that the success of any primitive algorithm (e.g. a controller, a vision system) or task can be measured by determining how reliable the algorithm or the task are, while simultaneously imposing cost constraint(s).

1.2 Problem Statement

The objective of the IM is to accomplish a goal dictated by an external *command.* However, the IM operates in a complex environment that disturbs the expected results of its actions. These disturbances result from incomplete environment modeling and unexpected events.

By assumption, the *environment* is modeled as a discrete-state stochastic process. At each instant in time, the environment can be in one of several states, drawn from a finite set. Every action of the IM over the environment generates some effect to which the environment responds. In most cases, the response denotes whether the action succeeded or failed. This response of a state of the environment to a given action is non-deterministic. The model assumes that some *probability of success* corresponds to each pair (action of HLSA, state of environment). This probability is also unknown to the IM. The environment is *non-stationary* if the probability of success of any action changes with time, and *stationary* otherwise.

The problem addressed by this work may thus be stated as:

A goal is dictated to a HGDIM by an external source (command). The HGDIM operates within a stochastic environment assumed to have the following characteristics:

- *it has a finite number of possible states. The transitions between states may depend on the current and/or past states, and they may be deterministic or stochastic;*

- *each state responds to an action of the IM with a* success *(1) or* failure *(0) signal;*
- *it has unknown probabilities of success for each pair (action of IM, state of the environment),*

The purpose is to design a reinforcement learning scheme which make the process of Decision Making converge to the selection w.p.1 *of the optimal actions at the two translation interfaces (Organization-to-Coordination and Coordination-to-Execution) of a HGDIM operating in the environment, for each of its states. Optimal actions are those which minimize a cost function at the corresponding interface: tasks in the top interface, primitive algorithms in the bottom interface. Defining this cost function is also part of the problem.*

1.3 Overview of Proposed Solution

In general terms, the HLSA and its Hierarchical Reinforcement Learning Scheme model the translation interfaces of a HGDIM as follows:

Learning: at the bottom level of the hierarchy, a cost function combining *reliability* and *computational cost* of the primitive algorithms is estimated from the success and failure responses of the environment to the application of one of these algorithms. This estimate is used by a *Reinforcement Learning Algorithm* to update the *subjective probabilities* of selecting the *primitive algorithms* capable of translating a *primitive event* at the bottom stage of the HLSA. Then, the cost function estimate is propagated to the top stage, where it is used by another *Reinforcement Learning Algorithm* to update the *subjective probabilities* of alternative *tasks* (sequences of *primitive events*) capable of translating a command sent to the machine.

Decision Making: Given a command, a task is selected by random decision at the top stage, based on the current task subjective probabilities for that command. At the bottom stage each of the primitive events composing the task is translated by a primitive algorithm selected by random decision, based on the current subjective probabilities of primitive algorithms.

Chapter 5 will show that the translation from commands to tasks is implemented by a *stochastic grammar*. Hence, the probabilities of selecting alternative productions of the grammar, not the probabilities of selecting tasks, are actually updated.

1.4 Organization and Terminology of the Thesis

The thesis is organized as follows:

Chapter 2 reviews the current literature in the areas of Architectures for Intelligent Machines, Learning Stochastic Automata and Stochastic Grammars, and Theory of Complexity;

Chapter 3 covers the basics of the theoretical background on Learning Stochastic Automata and defines relevant terminology;

Chapter 4 introduces the cost function based on reliability and computational cost of an algorithm. Several examples of application of the formalism are shown;

Chapter 5 introduces the new formulation of an Hierarchical Goal-Directed Intelligent Machine as a Hierarchical Learning Stochastic Automaton, with special emphasis on the feedback hierarchy. This is the main chapter of the thesis;

Chapter 6 studies the convergence rate of stochastic approximation algorithms and methods to accelerate it.

Chapter 7 describes two case studies based on the proposed formalism: one is concerned with Operations Management of a Glass Melting Furnace and the other with an Intelligent Robotic System.

Chapter 8 concludes the thesis and leaves some clues for future work.

Along the thesis, several abbreviations will be used quite often, such as:

- LSA for Learning Stochastic Automaton
- HLSA for Hierarchical LSA
- IM for Intelligent Machine
- HGDIM for Hierarchical Goal-Directed IM
- w.p.1 for With Probability One

Vectors will be underlined, such as in $\underline{x}$. Matrices will be denoted by capital letters. The context will distinguish them from some scalars also denoted by capital letters. Keywords will be italicized when they appear in the text for the first time and wherever else it is relevant.

1.5 Summary

In this chapter, the problem addressed by this thesis was stated and an overview of the proposed solution was introduced.

CHAPTER 2
Literature Review

In this chapter current and pioneer literature in the areas of Architectures for Intelligent Machines, Learning Stochastic Automata and Stochastic Grammars, and Theory of Complexity will be reviewed. These are the 3 research fields with major contributions to the theory developed in the following chapters.

2.1 Architectures for Intelligent Machines

The debate on Architectures for Intelligent Machines or Architectures for Intelligent Control Systems (AICS) is essentially divided today among those who propose a *Behavior-Based*, non-hierarchical solution [10] and a *Goal-Oriented* hierarchical architecture[3, 67]. In the the former formulation different agents compete to initially satisfy the basic needs of the machine (search for food, avoid obstacles), and after these are accomplished, to achieve more intelligent behavior, such as coordinating a few of those agents by assigning priorities. The latter distinguishes the levels of the hierarchy by the level of abstraction of the executed tasks, and a formulation close to the Theory of Control Systems is used to measure performance.

This distinction between goal-oriented, hierarchical architectures and behavior based, layered architectures, is somewhat artificial. While in the latter it is claimed that lower-level behaviors continue, even when momentarily subsumed by higher-level behaviors, these eventually take command in a hierarchical fashion. Similarly, there exist parallel distinct behaviors at each level of goal-oriented hierarchies [77].

The hierarchical architecture proposed by Saridis and Stephanou [72] for Intelligent Control of a prosthetic arm, analytically formulated by Saridis[70] and Saridis and Valavanis[73] is adopted in this work.

While these works put some emphasis on the Organization Level, Graham and

Saridis [23] proposed linguistic decision structures, called *Linguistic Decision Schemata*, used at all levels of the hierarchy, for the top-down translation of commands.

Wang and Saridis [93] refined this linguistic approach by proposing the implementation of the Coordination Level by a *Petri Net Transducer* (PNT). A PNT is composed of an Input Tape, a Petri Net Controller and an Output Tape. The Coordination Level is actually composed of a 2-stage hierarchy of PNTs. The top level implements the *Dispatcher* whose Output Tape is the input of several other PNTs implementing the Coordinators. The decision to fire an enabled transition of a PNT is based on the symbol currently read by the input head of the PNT. This symbol determines also the translation of the transition into some output symbol. The output symbols of the Coordinator PNTs represent low level algorithms which translate primitive events represented by the input symbols.

In Wang's work, there was a first attempt to address the feedback problem by updating the frequencies of success of each of the algorithms, but neither concrete measures such as reliability and complexity were mentioned nor a bottom-up propagation of such measures was envisaged. Furthermore, the entropy-based approach measures the uncertainty of translations at the Coordination Level, but does not take into account the uncertainty due to the reliability of the low level algorithms.

A major criticism of PNTs is that, despite the ingenious solution and the elegant formalism, the dispatcher seems difficult to design, even for problems of moderate complexity, since implicitly the different possible tasks coming from the Organization Level must be anticipated.

In recent work, Beard and Saridis [8] modified Wang's proposal in order to overcome this problem. A *Petri Net Transducer* or *Translator* (PNT) interprets its Input Tape as a string of some pre-defined language and translates it into a Hierarchical Petri Net. The macro-transitions of the top level Petri Net are translated into sub-Petri Nets representing the coordinators, whose transitions are in turn translated into low-level algorithms. A measure of structural cost for the alternative Petri Nets representing a task is also introduced.

Moed and Saridis [54] proposed a *Boltzmann Machine* to handle the combinatorial explosion associated to the Planning and Decision Making processes when different events are sequenced to form a task at the Organization Level. The convergence of the search for a task given a command is achieved using a *Modified Genetic Algorithm*. However, task translation is learned off-line and no update occurs after execution of a given task. Another potential drawback is that the design of the Boltzmann Machine, namely the required number of hidden units, is accomplished by heuristics.

McInroy and Saridis [50], and Musto and Saridis [56] introduced reliability and entropy-based criteria to choose the best algorithm among those capable of solving some problem, given a set of specifications for the problem, under some environment conditions. The environment conditions are implicitly assumed as stationary in both works, even though the latter relaxes the gaussian assumption for the noise in the former. Also, no learning of model parameters is involved. Actually, no learning at all is considered, since the main goal is to obtain a model for off-line selection of plans based on reliability and entropy.

The Execution Level has been implemented in CIRSSE, by utilizing a Space Truss Construction paradigm[14].

2.2 Learning Stochastic Automata and Grammars

The literature about learning automata models is vast and not limited to control applications. The first learning models were developed in mathematical psychology in the 1950s. However, this section will focus only on the most relevant works for control applications.

Tsetlin[83] introduced the use of deterministic automata operating in random environments to model learning. Later, Varshavskii and Vorontsova [89] introduced the concept of Stochastic Automata with Variable Structure, so called due to the update of the automata state probabilities along time.

Fu and his associates in the United States[20], Tsypkin and Pozniak in the Soviet Union[85], were among the first to propose Stochastic Automata models for Learning Control[17, 86].

The application of *linear reinforcement learning* to control systems was introduced by Waltz and Fu[91]. A seminal paper by Nikoliç and Fu[60] introduced the use of a performance function which is iteratively updated by a *stochastic approximation* algorithm and used by another stochastic approximation algorithm to learn the action probabilities of a controller. This approach is not based on Stochastic Automata, in a strict sense. McLaren[52] suggested the concept of "growing" automaton. An overview of the work of this group of researchers can be found in [66].

Fu and Booth[18] survey methods of Stochastic Grammar inference from a sample set of strings. The methods are generally valid for Context Free Grammars. Special emphasis is put on learning the productions probabilities when the following information is available: all the other grammar parameters, a sample set of strings, and the frequency of occurrence of a string in the sample set. A Maximum Likelihood Method which accomplishes this is described. The generation of alternative grammars is also referenced. Even though this is a time consuming process, one of the solutions described takes into account the complexity of the language generated to decide among grammars which fit equally well the sample set. This suggests a way of minimizing the structural cost referred in section 2.1.

Another school of research on the topic of Stochastic Automata is constituted by Narendra, Thathachar and their associates. Surveys of their work can be found in

[58, 59]. The latter is the most recent book on the subject, including more in-depth approaches to nonlinear reinforcement schemes and operation in non-stationary environments.

Lakshmivarahan and Thathachar[34] proved a necessary and sufficient condition for learning automata using a general nonlinear learning scheme to be absolutely expedient. The same authors[33] proposed a Bayesian technique to update the penalty probabilities of the environment, simultaneously leading to a smaller convergence time to the best action and providing a confidence level on the estimate of that action.

Recent work of these authors focus on Hierarchical Learning Stochastic Automata, where the actions of an automaton at one level represent automata at the level immediately below. The reward/penalty of the environment comes from either a single-teacher[78, 79] or multi-teacher[5].

The relation between stochastic approximation and stochastic automata is a subject of continuous debate. Narendra and Thathachar claim:
"*It is well known that stochastic approximation methods are applicable to parameter optimization problems, while the methods*" (of Narendra and associates) "*(...) are concerned with cases where probability distributions over finite action sets are updated*"[59].

However, Fu and Nikoliç introduced a general expression which encompasses linear reinforcement methods as a particular case of the stochastic approximation[21, 22]. These authors sought to conform original stochastic automata methods with stochastic approximation methods with the objective of proving convergence with probability 1 (w.p.1) of action probabilities. This convergence is not achieved by either linear or nonlinear reinforcement schemes not based on the stochastic approximation.

In the last few years, Barto, Sutton and their associates explored reinforcement learning solutions which associate these two schools [7, 74, 75, 76]. Some current work on *reactive agents*[41, 47] and *behavior-based agents* was triggered by [7], where the authors explore and compare the formulation of learning stochastic automata and supervised learning pattern classification to overcome the need to store a probability vector for each state of a non-stationary (multi-state stochastic) environment in a lookup-table. The use of this lookup-table is a memory-consuming procedure and slows the learning rate. In their formulation, the action probability vector is parameterized and a mapping from vectors of input features representing environmental states to the parameter is constructed. However, the algorithm proposed is limited to 2 actions, and this is the main drawback of the approach.

2.3 Theory of Complexity

The field of research on computational complexity has several branches. Most of the work focus on Combinatorial Complexity. The early work of Kolmogorov[30] introducing notions such as ϵ-entropy, Complexity and combinatorial foundations of Information Theory, has recent followers[39, 1] among Computer Science researchers. Zames[95] introduced Kolmogorov's ϵ-entropy in Control Theory. Most recently Tsitsiklis has developed work on Theory of Complexity in the context of Control Theory[84].

The so called Information-Based Complexity Theory was introduced in the early 1980s by Traub, Wasilkowsky and Wozniakowsky[81]. Information-Based Complexity differs from Combinatorial Complexity since in the former information is partial, noisy and costly, as opposed to complete, exact and free information in the latter. The information considered here is the *information contained* in the answers to questions about the problem element (see Chapter 4), not the *information content* of those answers, that is, the uncertainty about their correctness. This distinguishes the concept of information in Information-Based Theory of Complexity and the concepts of information and entropy in Shannon's Information Theory [80]. However, there is a link between how much information is needed to limit the uncertainty to a specified level and Kolmogorov's $\epsilon - entropy$.

2.4 Summary

This chapter reviewed the literature in the areas of Architectures for Intelligent Machines, Learning Stochastic Automata and Stochastic Grammars, and Theory of Complexity. The theory introduced in this thesis stands in the intersection of these three areas.

CHAPTER 3
Theoretical Background on Learning Stochastic Automata

This chapter provides the background needed to understand the theory of Learning Stochastic Automata used in the following chapters. Section 3.1 covers the basics of Learning Stochastic Automata with special emphasis on their generalized version, proposed by Fu and his associates, which uses a performance function and updates the decision probabilities by a stochastic approximation algorithm. Section 3.2 summarizes some concepts related to stochastic grammars which will be referenced in the sequel. Section 3.3 defines linguistically the task generation process in a HGDIM. Even though the previous work of Valavanis and Saridis[88] is partially followed here, the formulation has its own novelty and coherence, paving the way for the ideas presented in chapters 4 and 5.

3.1 Learning Stochastic Automata and Stochastic Approximation

Learning Stochastic Automata(LSA) [20, 59] have been suggested as appropriate solutions for the control of systems whose dynamics are completely or partially unknown and environments with unknown stochastic descriptions[17, 66] . Their application is particularly suited to levels where decision making is required, such as the Coordination and Organization Levels of Saridis' Intelligent Machine.

Some applications of LSAs are also usually associated to *performance-adaptive* methods, that is methods where the controller is modified structurally or parametrically along time, according to the evolution of the estimate of some performance function. This makes them even more relevant in this context.

Definition 3.1.1 *A LSA is defined by the quintuple* $\{Y, Q, U, F, G\}$*, where* $Y = \{0, 1\}$ *is the finite input set, with 1 representing a* reward *and 0 a* penalty*,* $Q =$

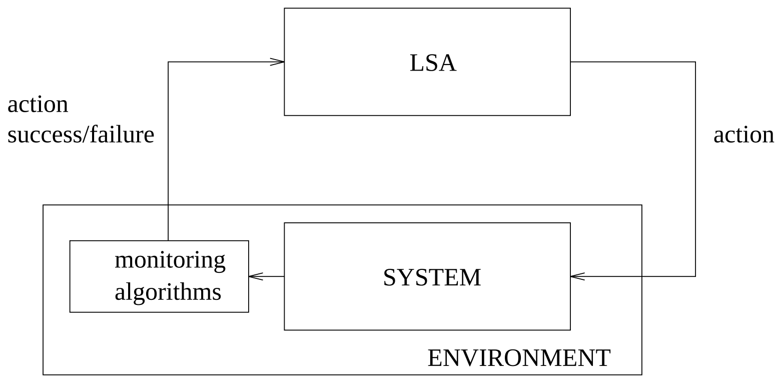

Figure 3.1: Closed loop between LSA and environment.

$\{q_1, \ldots, q_s\}$ *is a finite set of internal states and* $U = \{u_1, \ldots, u_s\}$ *is the output set or* action set. $F(P(n), y(n))$ *is called the updating or* reinforcement scheme *which generates* $P(n+1)$ *from* $P(n)$ *and* $y(n) \in Y$*, where* $P = \{p_1, \ldots, p_s\}$ *is the set of probabilities governing the (random) choice of state at each time instant, that is,* $P = P(n)$*. Finally,* G *is the output function* $G: Q \rightarrow U$*.* G *is stochastic in the general case, but with no loss of generality[59] it will be assumed to be represented by an identity matrix, i.e. each action is univocally and deterministically associated to a state.*

Any LSA interacts with the external world, usually denoted as the *environment* (see Figure 3.1). Every action of the LSA generates a response from the environment. In most cases, the response denotes whether the effect produced by the action was a *success* or a *failure*. In turn, the failure or success signal is used by the LSA to update its internal action probabilities according to the reinforcement scheme. An action probability is *rewarded* when applying the action over the environment results in a success or *penalized* when a failure occurs. This method is called *reinforcement learning*. These probabilities weight the random choice of the next state for the LSA, thus determining the next action.

In general, the response of the environment to a given action is non-deterministic. The interaction model assumes that some probability of success corresponds to each action. This probability is unknown to the LSA.

Definition 3.1.2 *A stochastic environment is defined by the quintuple* $\{U,\ X,\ Y,\ H,\ R\}$*, where* $U = \{u_1, \ldots, u_s\}$ *is the finite input set,* $X = \{x_1, \ldots, x_d\}$ *is a finite set of internal states and* $Y = \{0, 1\}$ *is the output set, where 1 represents a* success *and 0 a* failure*.* R *is a matrix whose general element* R_{ij} *is the probability of a success due to the application of input* j *over state* i*:*

$$R_{ij} \triangleq \Pr\{Y = 1 | x_i \in X,\ \ u_j \in U\}$$

R *determines the rate of failures and successes for each pair (input,state). If any of these rates change with time, the environment is non-stationary. Otherwise, the environment is stationary. Finally,* $H\ :\ X^r \rightarrow X$ *is the* state transition function *which generates* $x(n+1) \in X$ *from* $x(n), x(n-1), \ldots, x(n-(r-1)) \in X$*.*

The above definition of environment as a multi-state stochastic process or *chain* implies a non-stationary environment in the general case, because the probability of success depends on the current state. Only a single-state environment can be stationary. However, such an environment may be non-stationary if the probability of success of any of its actions continuously changes with time.

The non-stationarity of a multi-state environment may be overcome by generalizing the notion of LSA to a set of d sub-LSAs, where each sub-LSA is assigned to a different state $x_i \in X$ of the environment. This approach has two drawbacks: it may not be feasible if the actual number of states d of the environment is too large and it requires that the LSA is aware of the environment in which it operates[59]. Hence, methods capable of coping with non-stationary environments are worth studying. Analytic studies are known for specific models of non-stationary environments, such as a *Markovian Switching Environment*, where H depends only

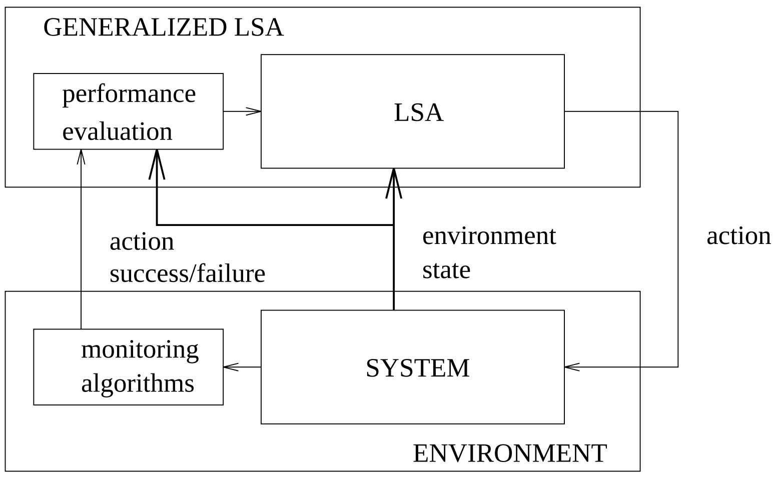

Figure 3.2: Closed loop between generalized LSA and environment.

on the current state ($r = 1$ in the above definition of environment). In this particular model of a stochastic environment, the overall system automaton + environment is equivalent to a homogeneous Markov chain, as shown in [59]. The case studies in this thesis propose an alternative strategy to deal with non-stationary environments.

Another extension of the basic concept of LSA was proposed by Fu[20], where the state probabilities are not rewarded or penalized directly. Instead, success and failure signals are used to update a *performance function* which is later used to update the probabilities (see Figure 3.2). There is a performance function for each pair (action, state of the environment).

When the LSA models a Hierarchical Controller, a *Hierarchical Learning Stochastic Automaton* (HLSA) must be used. At the bottom level of the hierarchy several LSAs interact directly with the environment. At the levels above, the action chosen by one of the LSAs corresponds to another LSA at the level immediately below. This is actually an advantage in terms of learning efficiency, since the high dimensionality of the decision space is overcome[78]. Some HLSAs were proposed which rely on this architecture. In some examples the reinforcement signal goes directly to all levels (single-teacher) [78], while in others there are different reinforcement signals for each level (multi-teacher)[5]. Chapter 5 introduces a different approach where reinforcement signals are propagated bottom-up.

The adaptation of the LSA to the state changes - with the consequent changes of the penalty probabilities - depends usually on the learning scheme used. Some schemes are capable of adaptive behavior in particular instances, such as the LRP scheme described by Narendra and Thathachar[59] or the Linear Reinforcement scheme of Waltz and Fu[91]. However, both schemes are not guaranteed to converge w.p.1 to the best action even if the environment is single-state.

An alternative consists of dealing with the large number of states of the environment in a hierarchical fashion. If a HLSA is considered, higher level states of the environment can be obtained as a composition of lower level states of the environment, thus reducing the actual number of options to explore. This subject will be further explored in section 5.5.

The behavior of an automaton can be measured to determine if its learning scheme leads to correct decisions after some interactions with the environment. Restricting the analysis to a single-state environment, one such measure is the *average reward* received by the automaton at instant n:

$$M(n) = \sum_{j=1}^{s} p_j(n) R_j \tag{3.1}$$

When all actions are chosen with equal probability, the average reward is denoted by M_0 and given by:

$$M_0 = \frac{1}{s} \sum_{j=1}^{s} R_j$$

Only a stochastic automaton with a learning scheme such that its average reward is greater than M_0 can be fairly called a Learning Stochastic Automaton as it is implicitly assumed in the above definition of LSA.

A LSA is called *expedient* if

$$\lim_{n\to\infty} E[M(n)] > M_0 \tag{3.2}$$

and *optimal* if

$$\lim_{n\to\infty} E[M(n)] = \max_{j=1,\ldots,s}\{R_j\} \tag{3.3}$$

Optimality implies that *asymptotically the action associated with the maximum reward probability is chosen with probability one*, that is, if $R_m = \max_{j=1,\ldots,s}\{R_j\}$,

$$\Pr\{\lim_{n\to\infty} p_m(n) = 1\} = 1 \tag{3.4}$$

For some learning schemes it can only be proved that in all stationary single-state random environments when $n \to \infty$, the LSA chooses with probability one an action whose reward probability belongs to a neighborhood ϵ of the maximum reward probability (ϵ-optimal LSA[58]). That is the case of the LRI (*Linear Reward Inaction*) scheme described by Narendra and his associates, which always converges to some action whose probability is one, but this action is not necessarily the optimal action. Optimality can only be obtained by suitable initial conditions for the probabilities and values of parameters of the learning scheme. Another learning scheme they describe, the LRP (*Linear Reward Penalty*) scheme, is only expedient, and its sequence of action probabilities distribution function converges to a distribution function at all points of continuity of the latter[58]. In these schemes, action probabilities are updated by a reinforcement scheme which uses the reinforcement signal from the environment directly.

Fu and his associates describe a different learning scheme that they apply to Learning Control Systems[17]. Fu's LSA updates first an estimate of performance for the current (LSA action, state of the environment) pair. This estimate is subsequently used to update action probabilities.

The *general performance function* Z_{ij} is defined as the mean value of the *instantaneous performance function* $y \in Y$ when the LSA action u_j is applied to the state of the environment x_i:

$$Z_{ij} = E[y|x_i, u_j]$$

y evaluates the response of the environment to a particular action (success or failure in the above definition of LSA).

The general performance function is estimated iteratively using a *stochastic approximation method*[20, 66]:

$$\hat{Z}_{ij}(n_{ij}+1) = \hat{Z}_{ij}(n_{ij}) + \gamma(n_{ij}+1)[y(n_{ij}+1) - \hat{Z}_{ij}(n_{ij})] \tag{3.5}$$

where $n_{ij} = 0, \ldots, \infty$ is the number of simultaneous occurrences of environmental state $x_i \in X$ and LSA action $u_j \in U$. $\hat{Z}_{ij}(n_{ij})$ denotes the n_{ij}th estimate of the mean value of the instantaneous performance function $y \in Y$. There are separate estimates for each pair (state of the environment, LSA action).

If

$$Z_{ij} < \infty, \quad E[y^2|x_i, u_j] < \infty \quad \hat{Z}_{ij}(0) < \infty$$

hold and

$$1 - \gamma(n_{ij}+1) > 0, \quad \sum_{n_{ij}=1}^{\infty} \gamma^2(n_{ij}+1) < \infty$$

$$\prod_{n_{ij}=1}^{\infty} (1 - \gamma(n_{ij}+1)) = 0, \quad i = 1, \ldots, d \quad j = 1, \ldots, s$$

Dvoretzky's conditions[15] are satisfied and the estimate converges w.p.1 to the actual value of the performance function, i. e.,

$$\Pr\{\lim_{n_{ij}\to\infty} \hat{Z}_{ij}(n_{ij}) = Z_{ij}, \ \forall i, j\} = 1$$

Action probabilities are updated by a reinforcement scheme also based on stochastic approximation:

$$p_{ij}(n_i+1) = p_{ij}(n_i) + \alpha(n_i+1)(\lambda_{ij}(n_i) - p_{ij}(n_i)) \tag{3.6}$$

where $n_i = \sum_j n_{ij}$ is the number of times any action has been applied to environmental state $x_i \in X$ so far, $0 \leq \lambda_{ij}(n_i) \leq 1$, $\sum_j \lambda_{ij}(n_i) = 1$, $i = 1, \ldots, d$ denotes states of the environment, and $j = 1, \ldots, s$ denotes LSA actions. Also,

$$1 - \alpha(n_i + 1) > 0, \quad \sum_{n_i=1}^{\infty} \alpha^2(n_i + 1) < \infty, \quad \prod_{n_i=1}^{\infty} (1 - \alpha(n_i + 1)) = 0, \quad i = 1, \ldots, d \tag{3.7}$$

and

$$p_{ij}(0) > 0, \quad \sum_{j=1}^{s} p_{ij}(0) = 1 \tag{3.8}$$

Given the estimates of the performance (or cost) function at each time instant, λ_{ij} is defined by

$$\lambda_{ij}(n_i) = \begin{cases} 1 & \text{if } \hat{Z}_{ij}(n_{ij}) = \min_k\{\hat{Z}_{ik}(n_{ik})\} \\ 0 & \text{if } \hat{Z}_{ij}(n_{ij}) \neq \min_k\{\hat{Z}_{ik}(n_{ik})\} \end{cases} \tag{3.9}$$

Defining also the *optimal action* as the action $u_m \in U$ such that

$$Z_{im} = \min_k\{Z_{ik}\} \quad i = 1, \ldots, d$$

the following Theorem is proved in [21]:

Theorem 3.1.1 *The necessary and sufficient condition for (3.6), (3.7), (3.8) and (3.9) to yield*

$$\Pr\{\lim_{n_i \to \infty} p_{im}(n_i) = 1\} = 1 \quad i = 1, \ldots, d$$

is that **for every sub-optimal action** $u_{j \neq m} \in U$

$$\sum_{n_i=1}^{\infty} \alpha(n_i) E[\lambda_{ij}(n_i) | y(1), \ldots, y(n_{ij})] < \infty, \quad i = 1, \ldots, d \quad j = 1, \ldots, s \quad j \neq m \tag{3.10}$$

Hence, Fu's generalized stochastic automaton can be made *optimal* in the sense of (3.3).

Essentially, the necessary and sufficient condition for optimality says that the estimates of the performance (or cost) function must converge faster than the action probabilities.

The convergence speed of stochastic approximation can be improved by the use of acceleration methods [22, 66, 27] and model-based initial estimates of reliability[20]. This will be discussed in chapter 6.

3.2 Stochastic Grammars

Grammars are usually employed to describe the syntax of languages or structural relations defining a pattern. They are useful in the context of HGDIMs to describe the constraints imposed to the ordering of events composing a task, as explained in section 3.3. In particular, *Stochastic Grammars* allow the assignment of probabilities to conflicting productions or rewrite rules. This turns out to be equivalent to the assignment of probabilities to the different strings of the generated *language*. The probabilities of the productions in each conflicting set can be learned by a LSA. Hence, stochastic grammars provide the means to learn the ordering of events composing a task, as will be shown in section 3.3 and Chapter 5.

Definition 3.2.1 *[18] A* stochastic grammar *is defined by the quintuple* $G = (V_T$, V_N, $\mathcal{R}$, $\mathcal{P}$, $S)$, *where*

- V_T *is a finite set of* terminal symbols*;*
- V_N *is a finite set of* nonterminal symbols*;*
- $\mathcal{R}$ *is a finite set of productions or rewrite rules;*
- $\mathcal{P}$ *is a finite set of probabilities that are assigned by a one to one mapping to the elements of* $\mathcal{R}$*;*
- S *is the start symbol.*

Only *stochastic regular grammars* will be considered here, that is stochastic grammars whose productions have the general syntax

$$A \to \alpha \ \text{ or } \ A \to \alpha B, \ \alpha \in V_T^*, \ A, B \in V_N$$

where the symbol to the left of the arrow is called *premise* while the term to the right of the arrow is the *consequent.* V_T^* denotes the set of all the possible strings composed by elements of V_T, including the null string.

The set of productions $\mathcal{R}$ can be partitioned into m disjoint subsets $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$, where m is the number of nonterminal symbols, $m = |V_N|$. In particular, $\mathcal{R}_i$ is the subset of productions with the same premise A_i, corresponding to the ith nonterminal symbol.

Correspondingly, the set of probabilities $\mathcal{P}$ can be partitioned into m disjoint subsets, where subset $\mathcal{P}_i$ contains the probabilities of the productions of $\mathcal{R}_i$.

A stochastic grammar is *proper* if

$$\sum_{k=1}^{m_i} p_{ik} = 1, \quad p_{ik} \in \mathcal{P}_i, \quad |\mathcal{P}_i| = m_i, \ i = 1, \ldots, m$$

To each string x of the language $L(G)$ generated by G corresponds a word function $f(x)$. If the grammar is unambiguous, that is, if there is only one leftmost derivation for each x,

$$f(x) = \prod_{k=1}^{K(x)} p(k, x), \ \forall x \in L(G)$$

where $K(x)$ represents the number of steps in the derivation of x, and $p(k, x)$ is the probability of the production used in the kth step of the derivation of x.

A language $L \subset V_T^*$, where V_T^* represents all strings of finite length composed by elements of V_T, including the null string ϵ, is called a *stochastic language* if there is a function $0 \leq f(x) \leq 1, \ \forall x \in L$, called *probabilistic word function*, $f(x): \ L \rightarrow \Re$, such that $\sum_{x \in L} f(x) = 1$.

Not all stochastic grammars generate stochastic languages. Some restrictions must be imposed to the stochastic grammar.

A stochastic grammar G is a *consistent grammar* iff the word function defined over $L(G)$ is a probabilistic word function, that is, iff $L(G)$ is a stochastic language.

If the grammar is proper, then it will be consistent.

3.3 Linguistic Formulation of Task Generation in HGDIMs

A *command* sent to the Intelligent Machine may be translated by more than one *task*, defined as an ordered sequence of *events*.

$T_i = \{t^i_1, \ldots, t^i_{l_i}\}$ is the set of *tasks* capable of implementing *command* c_i, $i = 1, \ldots, n_c$.

E is the set of all *primitive events*. $|E| = n_e$.

$E_i = \{e^i_1, \ldots, e_{m_i}{}^i\} \subset E$ is the set of *primitive events* compatible with *command* c_i. This definition reduces the search space when looking for events to compose a task. Notice that $E = \cup_{i=1}^{n_c} E_i$, but in general the sets E_i are not disjoint. That is, a primitive event may be compatible with two different commands.

We further associate to command i a *language* L_i whose set of *terminal symbols* is E_i. Task t^i_j is a string of L_i. Moreover,

$$T_i \subset L_i = E_i{}^+, \quad |T_i| = l_i$$

that is, task t^i_j, $j = 1, \ldots, l_i$ is one of the possible strings composed by elements of E_i, excluding the null string. The size of T_i depends on the constraints imposed by a given command to its compatible primitive events. These constraints are expressed by a *grammar* G_i which generates T_i and whose start symbol represents command c_i.

An *event* is a symbol representing a non-null string of primitive events. Events can be represented by non-terminal symbols of the grammar G_i which generates T_i.

A *primitive event* is an event which is no further decomposable. It represents a problem which can be solved by some algorithm. For each event there is at least one algorithm which translates the event, e. g., one algorithm which can solve the problem represented by the event. Primitive events are terminal symbols of the grammar G_i which generates T_i.

$A_k = \{a^k_1, \ldots, a_{n_k}{}^k\}$ is the set of alternative *algorithms* which may translate *primitive event* e_k, $k = 1, \ldots, n_e$.

3.4 Summary

This chapter presented the concepts and terminology considered relevant for the understanding of the following sections. Basic notions of Learning Stochastic Automata and Stochastic Grammars were summarized, and a linguistic formulation of an Intelligent Machine was introduced. The following chapters will put together all this theoretical background to introduce a new model of an IM with feedback from the environment.

CHAPTER 4
A Performance Measure Based on Computational Cost and Reliability

There are different options to reach a goal or a subgoal at the two highest levels of the IM: the **Organization Level** has to decide among different *tasks* capable of executing a given goal (*command*) sent to the machine; given the chosen task, composed by subgoals (*events*), the **Coordination Level** has to determine, for each event, the best among the set of *primitive algorithms* capable of solving each subgoal at the **Execution Level**. To compare the different alternatives at each level a *cost function* is necessary.

The different algorithms used at the Execution Level of an Intelligent Machine are frequently designed in order to meet a set of *specifications* or, without loss of generality, in order to keep the error between the actual and desired values of a set of variables below some desired *accuracy* ϵ.

The uncertainty involved in the design of these algorithms is mostly due to approximate or incomplete modeling and statistical fluctuations around nominal parameters. Previous work in this area [49, 56] models the uncertainty associated to the different algorithms using the concepts of *reliability* and *entropy*. It describes algorithm selection techniques, based on entropy, which will choose the most reliable from a set of different algorithms capable of solving some specific problem. However, the most reliable algorithm may have a non feasible computational cost, in terms of the time it takes to complete, the amount of memory it uses, or the number of resources (e.g. processors) required. No attempt is made in this work to deal with this problem, with the exception of plan execution time, modeled as a specification by McInroy and Saridis[50].

Thus, it makes sense to think of a selection technique which includes both reliability and computational cost, but first the two measures must be coherently defined. Information-Based Theory of Complexity provides some assistance to solve this problem.

The *computational complexity* of a *problem* is defined by Traub *et al* as "*its intrinsic difficulty as measured by the time, space or other quantity required for its solution*"[80].

More formally this is equivalent to the cost of the optimal *algorithm* (in the sense of computational cost) for the solution of the *problem.*

Two main kinds of computational complexity may be categorized into:

- **information-based complexity**, when information about the problem is **partial** (e.g. aliasing - more than one signal has the same values at the sampling instants), **noisy** (e.g. the samples are corrupted by noise) and **costly** (e.g. the smaller the sampling time the more costly is the sampling operation);
- **combinatorial complexity**, when information about the problem is **complete** (e.g.: all distances in a robot-in-a-maze problem are given), **exact** (the distances are assumed to be error-free) and **free** (there is no charge to know the distances).

In the sequel, the focus will be on Information-Based Complexity. Problems associated to Intelligent Machines, whether they consist of position or force controllers, path or trajectory planners, robotic vision algorithms or others, deal with information of all kinds, and this information is often partial, noisy and costly. Moreover, this work deals with strongly uncertain environments, and it has been pointed out before that reducing the degree of uncertainty in controlling those environments is the goal.

This chapter introduces a cost function combining reliability and computational cost (cost for short, in the sequel) of an algorithm, based on a coherent definition of the two. The link between reliability and cost is the assumption that all algorithms are designed to meet an *error specification* for the problem they can solve. Given some desired reliability for the problem, the cost of obtaining that reliability can be determined for each of the algorithms, according to the cost measure defined (number of operations, elapsed CPU-time, memory used) for the problem. Conversely, if the cost measure is fixed at different values for the different algorithms, this will correspond to different reliabilities for each of them.

This formulation has been introduced by Lima and Saridis in [42, 46] and it is based on the Theory of Information-Based Complexity[81].

The next section summarizes the general formulation of information-based complexity. The original theory is adapted to this formulation when needed. The second section coherently defines cost and reliability. In the third section the cost function and the equations to propagate its value bottom-up through the hierarchy are introduced. Finally, several examples of application of the formalism to Robotic Systems are described in detail in the last section.

4.1 Information-Based Computational Cost of a Problem

4.1.1 Problem Formulation

For each $f \in F$, where F is a set of *problem elements*, it is desired to compute an approximation $U(f)$ of $S(f)$, where $S : F \rightarrow G$ is called a *problem solution* and G is a normed linear space over the scalar field of real or complex numbers. To measure the distance between $S(f)$ and $U(f)$ an absolute error criterion, $\|S(f) - U(f)\|$ is used.

$U(f)$ is an *ϵ-approximation* of $S(f)$ iff $\|S(f) - U(f)\| \leq \epsilon \geq 0$. The original theory establishes three different settings for the error of the approximation $U(f)$ of $S(f)$: *worst-case, average* and *probabilistic*[81]. In the first two settings, $U(f)$ must be an *ϵ-approximation* of $S(f)$ in the worst-case specification error ($\|S(f) - U(f)\|$) or for the average specification error, respectively. In the probabilistic setting, which will be used in the sequel, the specification error is required to be below ϵ except in a subset of G with a small measure.

4.1.2 Information

It is assumed that the only initial existing knowledge about f is that it belongs to the set F, and also that more knowledge about f may be gathered using computations of the form

$$L(f), \ L \ : \ F \rightarrow H$$

for some set H.

L must belong to the class Λ of permissible information operations (*oracles*), that is $L \in \Lambda$ iff L can be computed for each $f \in F$.

H may assume several different forms. For example, it may either be the set $\{0,1\}$ of answers to a question like "what is the intensity value of pixel (i,j) in some black-and-white image?" or the set of real numbers when Λ is a collection of a function and its derivative values at some point x, $L_i(f) = f^{(i)}(x), \ 0 \leq i \leq r$.

The information $\mathcal{I}(f)$ is then defined as

$$\mathcal{I}(f) = (L_1(f), L_2(f), \ldots, L_n(f))^T, \ \forall f \in F.$$

.

Finally, $U(f,\phi) = \phi(\mathcal{I}(f))$ where $\phi(\mathcal{I}(f)) \in G$ is an *algorithm* that computes an approximation of $S(f)$ given the information $\mathcal{I}(f)$.

4.1.3 Model of Computation

The initial assumptions are:

- either a sequential or parallel model of computation is assumed;
- there is a charge for each information operation;

- all information and combinatorial operations are performed with infinite precision and finite cost.

The model postulates a constant cost c for each information operation $L(f) \in \Lambda$ and unit cost for each combinatorial operation performed by ϕ over $\mathcal{I}(f)$.

The *cost* of an algorithm ϕ has two components:

$$\text{cost}(\phi, f) = c_i(\mathcal{I}(f), f) + c_p(\phi, \mathcal{I}(f)) \tag{4.1}$$

where c_i is the cost of getting information about f needed by algorithm ϕ, and c_p is the combinatorial cost of processing that information by algorithm ϕ. Given the above, $c_i(\mathcal{I}(f), f) \geq c|\mathcal{I}(f)|$, where $|\mathcal{I}(f)|$ denotes the cardinality of $\mathcal{I}(f)$, that is, the number of information operations. The term c_i is inherent to information-based complexity. Information is gathered to reduce uncertainty. c_p would be the only term in the absence of uncertainty.

Under the probabilistic setting we control the error of estimating $S(f)$ by $U(f, \phi)$ (the result of algorithm ϕ), keeping it below ϵ, except in a subset of G with measure $\delta \in [0, 1]$

Given ϵ and δ, the cost of an algorithm is obtained for the most unfavorable problem element f whose approximated solution $U(f)$ still belongs to the subset of G with measure $1 - \delta$:

$$\begin{aligned} f^* &= \arg \inf_{f \in F} \{\Pr\{\|S(f) - U(f, \phi)\| < \epsilon\} \ni \Pr\{\|S(f) - U(f, \phi)\| < \epsilon\} \geq 1 - \delta\} \\ \text{cost}(\phi) &= \text{cost}(\phi, f^*) \end{aligned} \tag{4.2}$$

For example, N image frames or more need to be averaged to increase to a certain value the probability that the error of locating an object in a noisy image is below ϵ. Every image resulting from the average of a different number frames is a problem element. If the cost of processing that information is not considered, the overall cost will be equal to c_i and proportional to the number of averaged frames.

Among the number of image frames which have to be averaged, N corresponds to the worst-case specification error. A greater number of averages will decrease the error probability, while a smaller number will push the corresponding approximated problem solution to the subset of G with measure δ, for which $\Pr\{\|S(f)-U(f,\phi)\| < \epsilon\} < 1-\delta$.

4.2 Coherent Definition of Reliability and Complexity

In order to coherently define cost and reliability for a given problem, the desired accuracy ϵ of the problem specification must be the same in both definitions.

Given some problem, an algorithm capable of solving it with the required accuracy may not exist, due to the inherent uncertainty of the problem. This uncertainty is measured in Information-Based Theory of Complexity by the problem *radius of information*, which resembles the selection of feasible plans proposed by McInroy and Saridis[49], based on the comparison of plan and specification entropies.

Assuming that the desired accuracy is greater than the radius of information, *ϵ-cost* (*cost* for short) of a problem is defined here as the minimal cost among the set Φ_{feas} of all available and feasible algorithms which solve the problem with error defined in the probabilistic sense:

$$\epsilon\text{-cost} = \inf_{\phi \in \Phi_{feas}} \{\text{cost}(\phi)\} \tag{4.3}$$

Suppose now that in (4.2) $S(f)$ is a vector of specifications for a given problem. The problem solution $S(f)$ is for example the desired overshoot of a control algorithm implementing a move robot event, and the problem element f is the output signal used to compute $U(f,\phi)$, the problem approximation.

Definition 4.2.1 *Given some desired specification accuracy ϵ, and a problem element $f \in F$, reliability of an algorithm ϕ is defined as:*

$$R(\phi, f) \quad = \quad \Pr\{\text{specifications met}\}$$

$$\begin{aligned} &= \Pr\{\text{specification error} < \epsilon\} \\ &= \Pr\{\|S(f) - U(f, \phi)\| < \epsilon\} \qquad (4.4) \end{aligned}$$

Now, making $R_d = 1 - \delta$ when determining the cost of ϕ by equation (4.2), the definition of cost is obtained.

Definition 4.2.2 *Given some desired lower bound for the reliability R_d, cost of an algorithm ϕ is defined as*

$$f^* = \arg \inf_{f \in F} \{R(\phi, f) \ni R(\phi, f) \geq R_d\} \qquad (4.5)$$

$$C(\phi) = \text{cost}(\phi, f^*) \qquad (4.6)$$

that is, among all $f \in F$ capable of keeping the specification error for algorithm ϕ below ϵ with reliability at least R_d, the one leading to the worst-case, i.e. the f leading to the larger probability of error, is picked. Here and henceforth, the reliability will be denoted as $R(\phi) = R(\phi, f^*)$.

Notice that $U(f, \phi)$ is a random variable due to the noisy measures of f and incompleteness of the model. R_d is a variable which may be used to help the design of an algorithm such that some reliability is achieved. However, the algorithm is usually designed to satisfy the requirements on accuracy ϵ and then tested to check its reliability. R_d could also be improved by increasing ϵ (relaxing constraint on specification error), but that is **not** what is assumed here. Once ϵ is fixed, there are two possibilities:

- The different algorithms for the problem are designed to meet the accuracy specification assuming no uncertainty. Their cost is determined by the number $|\mathcal{I}(f)|$ of information operations plus the combinatorial complexity of processing such information. In this case, reliability is determined by (4.4) and R_d is the reliability lower bound;

- The different algorithms for the problem are designed given the accuracy ϵ and reliability R_d specifications. In this case, the alternative algorithms for a problem typically differ by the number of information operations they need. Hence, the cost is determined from (4.2), given ϵ and R_d.

The main differences between the definition of cost of an algorithm proposed here and the definition of Information Based Complexity in the probabilistic case will now be explained. The analysis provides a better understanding of the coherent definition of reliability and cost.

Let $\delta \in [0, 1]$. The *probabilistic error* associated to algorithm ϕ is defined in the context of Information Based Complexity by

$$e(\phi) = \inf_{A}\{ \sup_{f \in F-A} \|S(f) - U(f, \phi)\| \ : \ p(A) \leq \delta\}$$

In other words: given a set A of measure less than or equal to δ, the supremum of the approximation error among all $f \in F - A$ is sought. Then the error is minimized by seeking the set A (among all sets with $p(A) \leq \delta$) leading to the infimum of the suprema of the errors.

The *probabilistic cost* of algorithm ϕ, under the worst case setting for the cost, is given by

$$\text{cost}(\phi) = \sup_{f \in F} \text{cost}(\phi, f)$$

that is, the cost of the algorithm does **not** depend on the specification ϵ.

The complexity of the problem depends on ϵ and it is defined as the minimal cost among all ϕ with probabilistic error at most ϵ:

$$\epsilon\text{-comp} = \inf_{\phi}\{\text{cost}(\phi) \ : \ e(\phi) \leq \epsilon\}$$

The algorithm ϕ^* that achieves the minimal cost is called an *optimal algorithm.*

This assures that the largest error among all realizations $f \in F - A$ will not exceed ϵ. The cost is determined for the worst case among these realizations.

However, with probability less than or equal to δ, there exists a set A which is not checked, where some realization may lead to a larger error. If the algorithm of minimum cost for $f \in F - A$ happens to have an actual error greater than ϵ when all $f \in F$ are considered, the increase in the actual error leads to an increase of complexity, because some other algorithm ϕ' of larger cost will replace the previous optimal algorithm.

Equation (4.2) states that the cost is obtained for the worst-case f such that the reliability is greater than or equal to $1 - \delta$. The algorithm may fail to meet its specifications with probability δ. By Definition 4.2.2, among the $f \in F$ which achieve this reliability, the one of largest probability of error which still satisfies $R(\phi) \geq R_d$ leads to the cost ϕ. That is, the probability distribution depends in this case on a function of f, and not on f itself. As a consequence of this, the cost of an algorithm depends on the accuracy ϵ and the reliability, and so does the complexity of the problem.

In summary, the probabilistic setting of Information Based Complexity assumes uncertainty in the investigation of all possible situations an algorithm is applied to, while the probabilistic setting used in this thesis assumes uncertainty in the success of an algorithm, in the worst case for the error probability, among all possible situations.

4.3 A Cost Function for Intelligent Machines

The coherent definition of reliability and complexity introduced in the previous section allows the definition of a cost function combining the two, assuming that each algorithm is designed to meet a set of specifications.

Definition 4.3.1 *A* **Cost Function** *valid at all levels of a Hierarchical Intelligent Machine is defined by:*

$$J = 1 - R + \rho C \tag{4.7}$$

where R is the reliability, C the cost and ρ a normalizing factor such that $\rho C \in [0,1]$.

In general ρ will be such that the cost does not overwhelm the reliability when searching for the optimal action. Examples of ρ are $\rho = \frac{1}{\max_{a\in A} C(a)}$ or $\rho = \frac{1}{\sum_{a\in A} C(a)}$, where A is the set of algorithms capable of solving a problem.

Definition 4.3.2 *The optimization problem described in this thesis consists of finding the task and primitive algorithms which minimize (4.7) for a given command issued to the machine and a given state of the environment.*

Equation (4.7) applies to all levels of the HGDIM, i.e., the performance of an algorithm, primitive event or task can evaluated by (4.7) if the cost and reliability are appropriately propagated bottom-up through the hierarchy.

Recall from the previous chapter that a task t is composed by several events $e_k \in E$, occurring either sequentially or in parallel. For each event e_k there exist a set of alternative primitive algorithms A_k capable of solving the problem represented by the event, for $k = 1, \ldots, n_e$, where n_e is the total number of primitive events.

The propagation equations are:

Definition 4.3.3 *The* **Cost of event** $e_k \in E$ *is the minimum cost among all algorithms translating the event:*

$$C(e_k) \triangleq \min_{a\in A_k}\{C(a)\} \quad k = 1, \ldots, n_e \tag{4.8}$$

The **Action probability** p_a **of algorithm** $a \in A_k$ is the current probability of a being applied. A probability density function is defined over the discrete algorithm space A_k. Its purpose, discussed later in this thesis, is to help a learning algorithm converging to the algorithm which minimizes the cost function J.

Definition 4.3.4 *The* **Reliability of event** e_k *is the average reliability among all algorithms translating the event:*

$$R(e_k) \triangleq \sum_{a\in A_k} p_a R(a) \quad k = 1, \ldots, n_e \tag{4.9}$$

If the cost function is to be used for designing purposes only and no learning is involved, an alternative definition which does not use action probabilities is:

Definition 4.3.5 *The* **Reliability of event** e_k *is the maximum reliability among all algorithms translating the event:*

$$R(e_k) \triangleq \max_{a \in A_k} R(a) \quad k = 1, \ldots, n_e \tag{4.10}$$

Definition 4.3.6 *The* **cost of parallel execution of events** e_1, e_2 *is*

$$C(e_1 // e_2) \triangleq \max_{e_1, e_2 \in E} \{C(e_1), C(e_2)\} \tag{4.11}$$

while the **cost of** n **events executed in series** *is*

$$C(e_1 | \ldots | e_n) \triangleq \frac{1}{n} \sum_{i=1}^{n} C(e_i), \quad e_1, \ldots, e_n \in E \tag{4.12}$$

The mean in equation (4.12) intends to keep the cost in the interval $[0, 1]$.

The successive application of these rules leads to the **cost of a task**, $C(t)$. An additional *structural cost* may be added, for example as suggested by Beard and Saridis (1993)[8].

The parallel execution of events is not logically parallel from the reliability point of view. In fact, **all** events must be successful to complete a task. Hence,

Definition 4.3.7 *The* **reliability of task** t *is*

$$R(t) \triangleq \prod_{e_k \in \mathcal{E}_t} R(e_k) \tag{4.13}$$

where $\mathcal{E}_t \subset E$ is the set of events composing the task.

In chapter 5 the propagation equations will be rewritten to include reliabilities conditioned by the state of the environment, and command cost and reliability. The above definitions apply to the particular case of a single-state environment and were included here to help understanding how the cost function can be used at all levels of an Intelligent Machine.

4.4 Applications to Robotic Systems

Tasks implemented by Intelligent Robotic Systems may generally be decomposed on primitive events. Among these the most typical are perhaps Move Robot, Locate Object, Plan Path, Plan Trajectory, Grasp Object. Algorithms capable of solving these problems belong to the areas of Motion Control, Computer Vision, and Path or Trajectory Planning. In this section it will be shown how the performance of some of these algorithms may be computed under the paradigm just formulated. The following three subsections cover examples of optimal pose (position + orientation) control of manipulators, compliance control using a position accommodation technique and image processing. Emphasis was put on cost measures other than execution or computation time, to enhance the flexibility of the definition.

4.4.1 Pose Control

The dynamics of a n-degree of freedom robot manipulator can be expressed by the following compact form of Euler-Lagrange's equations of motion:

$$D(\underline{\theta})\ddot{\underline{\theta}} + \underline{NL}(\underline{\theta}, \dot{\underline{\theta}}) = \underline{u} \tag{4.14}$$

where $\underline{\theta} \in \Re^n$ is the joint angles vector, $\underline{u} \in \Re^n$ is the control torques vector, $D(\underline{\theta}) : \Re^n \rightarrow \Re^{n\mathrm{X}n}$ is the inertia matrix, and $\underline{NL}(\underline{\theta}, \dot{\underline{\theta}}) : \Re^n \mathrm{x} \Re^n \rightarrow \Re^n$ is the vector representing nonlinear coupling of Coriolis, centrifugal, gravity and friction torques. Luo and Saridis (1985)[48] formulated the optimal control solution for the problem of making the manipulator track a desired trajectory. They identified the system state with $\underline{x}(t) = (\underline{\theta}(t)\ \dot{\underline{\theta}}(t))^T$ and suggested the performance index

$$J(\underline{u}) = \frac{1}{2}\underline{e}^T(t_f)G\underline{e}(t_f) + \frac{1}{2}\int_{t_0}^{t_f}[\underline{e}^T(t)Q\underline{e}(t) + \dot{\underline{e}}^T(t)S\dot{\underline{e}}(t)]dt \tag{4.15}$$

where $S = \begin{pmatrix} 0 & 0 \\ 0 & S_0 \end{pmatrix}$, G is a $2n$x$2n$ and S_0 a nxn real symmetric, positive definite matrix, Q is a real non-negative $2n$x$2n$ matrix, $\underline{e}(t) = \underline{x}_d(t) - \underline{x}(t)$ and $\underline{x}_d(t) =$

$(\underline{\theta}_d(t)\dot{\underline{\theta}}_d(t))^T$ is the desired state vector. When $t_f \to \infty$, the control law reduces to

$$\underline{u}^* = D(\underline{\theta})\{\ddot{\underline{\theta}}_d(t) + K_p[\underline{\theta}_d(t) - \underline{\theta}(t)] + K_v[\dot{\underline{\theta}}_d(t) - \dot{\underline{\theta}}(t)]\} + \underline{NL}(\underline{\theta}, \dot{\underline{\theta}}) \tag{4.16}$$

which has the same form of the *Computed Torque Method*, with $K_p = S_0^{-1}P_{12}$ and $K_v = S_0^{-1}P_{22}$. $P = \begin{pmatrix} P_{11} & P_{12} \\ P_{12} & P_{22} \end{pmatrix}$ is the solution of a continuous algebraic Riccati equation.

Given the optimal control law, the closed loop state space model is

$$\dot{\underline{x}}(t) = A_{cl}\underline{x}(t) + B_{cl}\underline{u}_d(t) \tag{4.17}$$

where $A_{cl} = \begin{pmatrix} 0 & I \\ -K_p & -K_v \end{pmatrix}$, $B_{cl} = \begin{pmatrix} 0 & 0 & 0 \\ K_p & K_v & I \end{pmatrix}$, and $\underline{u}_d = \begin{pmatrix} \underline{\theta}_d \\ \dot{\underline{\theta}}_d \\ \ddot{\underline{\theta}}_d \end{pmatrix}$.

I and 0 denote nxn identity and zeros matrices, respectively. The model can further be discretized by some suitable method, and a discrete time state space model will be obtained

$$\underline{x}((k+1)T_s) = A_{dcl}\underline{x}(kT_s) + B_{dcl}\underline{u}_d(kT_s) \tag{4.18}$$

if the sampling period is T_s.

In this development it has been assumed:

1. Perfect cancellation of the non-linear terms;

2. Non-noisy measurements;

3. Complete information about the state.

However, assumption 3 can be kept but 1 and 2 may be relaxed by modeling the resultant perturbations as zero mean gaussian noise. Then a new discrete state model is obtained:

$$\underline{x}((k+1)T_s) = A_{dcl}\underline{x}(kT_s) + B_{dcl}\underline{u}_d(kT_s) + D\underline{v}(kT_s) \tag{4.19}$$

where $\underline{v}$ is a gaussian noise vector with $E[\underline{v}(kT_s)] = 0,\ E[\underline{v}(kT_s)\underline{v}(kT_s)^T] = C_v$.

The performance index has to be modified when the noise is actually added to the open loop system, and it becomes $I(\underline{u}) = E[J(\underline{u})]$.

For this *pose control problem* (event move robot) the cost of an algorithm ϕ solving the problem will be the optimal value of I:

$$C(\phi) = I(\underline{u}^*) = \underline{e}(0)^T P \underline{e}(0) + \sum_{k=1}^{N} \operatorname{tr}(PDC_vD^T) \tag{4.20}$$

where P is the solution of a discrete algebraic Riccati equation (Lewis, 1986 [38]), and N the number of samples in the trajectory.

A lower bound for the Reliability can be obtained based on a method described by McInroy and Saridis (1994) [49], when the specifications are quadratic in the tracking error $\underline{e}(kT_s)$:

$$\underline{e}(kT_s)^T Q_s \underline{e}(kT_s) \le \epsilon,\ k = 1, \ldots, N,\ Q_s \ge 0 \tag{4.21}$$

If

$$C_e^{-1}(kT_s) - Q_s(kT_s) \ge 0, \forall k = 1, \ldots, N \tag{4.22}$$

then

$$R(\phi) \ge [\chi_d^2(\epsilon)]^N \tag{4.23}$$

where χ_d^2 is a chi-square distribution with d degrees of freedom, $C_e(kT_s)$ is the covariance of the tracking error, N the number of points the specifications are concerned with, and d the dimension of the state vector ($d = 2n$ for a n-degree of freedom manipulator). $C_e(kT_s)$ can be determined by solving the difference equation

$$C_e((k+1)T_s) = A_{dcl} C_e(kT_s) A_{dcl}^T + DC_v(kT_s)D^T \tag{4.24}$$

Given Q_s and ϵ, the reliability lower bound is given by (4.23) for all different C_e which satisfy (4.22). The value of C_e depends on A_{dcl} which in turn is a function of the weighting matrices Q, S, G in the performance index. Hence, for different lower bound reliabilities, different Costs C will be obtained, and the performance function $J = 1 - R + \rho C$ helps deciding among different optimal algorithms resulting from different choices of Q, S, G.

When a Computed Torque algorithm is used, K_p and K_v are usually made diagonal ($K_p = k_p I$ and $K_v = k_v I$) and k_p, k_v are dimensioned to obtain desired specifications for each of n decoupled control loops. For each loop, and assuming perfect cancellation of the non-linear terms,

$$w_n = \sqrt{k_p} \tag{4.25}$$

$$\xi = \frac{k_v}{2\sqrt{k_p}} \tag{4.26}$$

where w_n is the *natural frequency* and ξ is the *damping factor*. Notice that k_p and k_v are scalars.

A natural approach to the translation of a Move Manipulator event will be to select a set of Computed Torque algorithms with different k_p and k_v (thus different w_n and ξ), and determine their costs and reliability. An expression to compute the cost will be derived next.

Equation (4.20) is a truncated version of the actual expression for the cost of the discretized system under the assumptions of measurement noise and $t_f \to \infty$:

$$C = I(\underline{u}^*) = \underline{e}(0)^T P \underline{e}(0) + \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \mathrm{tr}(PDC_vD^T) \tag{4.27}$$

where $N = t_f/T_s$, from (4.20). The cost is obtained for the number of steps taken by the actual displacement of the manipulator from the start to the end point only, but P is the steady-state solution of the Riccati Equation (when $N \to \infty$).

Now, making $\underline{e}(0) = 0$, $D = I$ and $C_v = \begin{pmatrix} \sigma_p^2 I & 0 \\ 0 & \sigma_v^2 I \end{pmatrix}$ where σ_p^2 is the variance of position noise and σ_v^2 the variance of velocity (or encoders) noise, the cost simplifies to

$$C = \mathrm{tr}(PC_v)N \tag{4.28}$$

In this expression, the sub-matrices of P are not known. However, the expressions for P_{12} and P_{22} may be simplified because K_p and K_v are diagonal matrices:

$$\begin{aligned} P_{22} &= S_0 k_v I && (4.29) \\ P_{12} &= S_0 k_p I && (4.30) \end{aligned}$$

The steady-state Riccati Equation is

$$Q - PSP + PF + F^T P = 0 \tag{4.31}$$

where $Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{12} & Q_{22} \end{pmatrix}$.

Assuming different S_0 and Q will lead to different solutions for P. Solving (4.31) for P_{11} given (4.29) and (4.30), one obtains

$$P_{11} = -Q_{12} + S_0 k_v k_p I \tag{4.32}$$

One possible solution used here consists of making

$$\begin{aligned} S_0 &= I \\ Q_{12} &= (k_p k_v - 1)I \end{aligned}$$

From these assignments one obtains

$$\begin{aligned} P_{11} &= I \\ P_{22} &= k_v I \\ P_{12} &= k_p I \\ Q_{11} &= {k_p}^2 I \\ Q_{22} &= ({k_v}^2 - 2k_p)I \\ Q_{12} &= (k_p k_v - 1)I && (4.33) \end{aligned}$$

and the cost comes, for a n degree-of-freedom manipulator

$$C = n(\sigma_p^2 + k_v \sigma_v^2)N \tag{4.34}$$

If encoder noise can be ignored, $\sigma_v^2 = 0$ and the cost is proportional to the time taken by the movement.

Another alternative is to make

$$\begin{aligned} S_0 &= I \\ Q_{12} &= 0 \end{aligned}$$

In this case, the cost is

$$C = n(k_p k_v \sigma_p^2 + k_v \sigma_v^2)N \tag{4.35}$$

and it depends on k_p and k_v also.

Other combinations of Q and S_0 might have been used.

The following instantiations of the definitions above for this particular example summarize and clarify the application of the formalism:

- problem element $\underline{f} = (\underline{x}\ \underline{x}_d)$
- problem solution $\underline{S}(\underline{f}) = \underline{x}_d$
- solution approximation $\underline{U}(\underline{f}, \phi) = \underline{x}$, as obtained by algorithm ϕ (includes noise $D\underline{v}$).
- algorithm $\phi = \phi(Q, S, G) = \underline{u}^*(Q, S, G)$

The performance function associated to the algorithms balances the penalty of error and cost of control (by penalizing joint accelerations) to track a given trajectory (joint positions, velocities and accelerations) and the reduction of uncertainty due to measurement noise.

4.4.2 Compliance Control

The robot comes in contact with the environment while performing many useful tasks. During the execution of these tasks the robot controller should control the forces exerted by the robot in relation with the motion of the end-effector. Thus the robot may be required to exhibit a particular functional relation between the force it exerts and the displacement that results. For contact tasks the desired relationship is an impedance. Impedance control involves issuing a position command and assigning a relationship between the interaction forces and deviations from the desired position command. Thus, impedance control consists of a position control loop with the assigned impedance determining the stiffness of the manipulator[24]. This type of compliance control is called *Position Accommodation Control*[65].

Suppose the tip of a 6 degree-of-freedom manipulator is required to behave as a *Mass, Spring and Damper* system. Let the [6x1] vectors x_0 be the nominal end-effector trajectory and x be the actual end-effector pose (cartesian position + orientation). Let f be the forces and torques on the manipulator due to contact with the environment.

$$\underline{f} = K(\underline{x} - \underline{x}_0) + B(\dot{\underline{x}} - \dot{\underline{x}_0}) + J(\ddot{\underline{x}} - \ddot{\underline{x}_0}) \tag{4.36}$$

Equation (4.36) represents a relationship between the force at the end-effector and motion about a nominal trajectory. If $\underline{x} = \underline{x}_0$ the force $\underline{f}$ is zero. Thus $\underline{x}_0$ can be considered the non-contact trajectory. The choice of the [6x6] matrices K, B, and J depend upon the response desired from the system. Their values will also determine the cost and reliability of compliance control algorithms.

Suppose a manipulator has to grasp some object using impedance control. After getting to a position above the object with the required tool pose and xy position, the manipulator tip (tool) must approach the object with a vertical downward movement along the z axis. Once the object is reached, the manipulator will try to grasp it after some desired force in the positive z direction is obtained or the pre-established duration time for the movement expires, whichever occurs first. In this study, compliance is assumed to work for all other components of $\underline{x}$, and K, B and J are scalars.

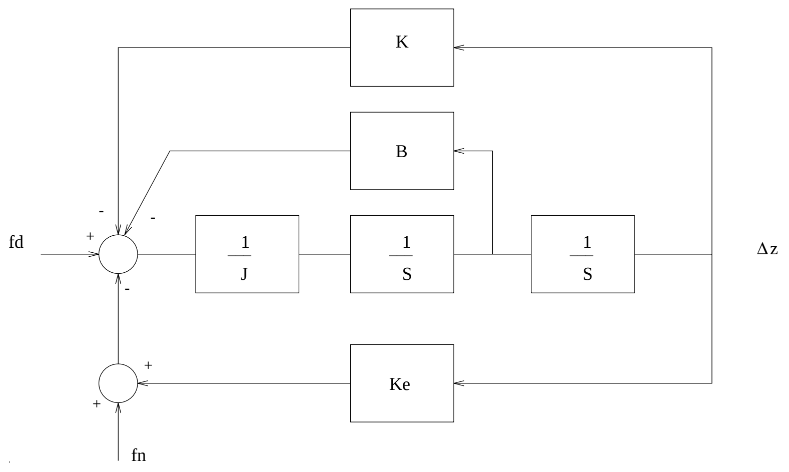

Figure 4.1: Continuous Mass, Spring and Damper block diagram.

If the downward movement is exclusively due to a desired force f_d, the closed loop manipulator-environment can be roughly modeled as in Figure 4.1. The environment is modeled as a spring of constant K_e and errors from the manipulator position controller are ignored. $K_e = 0$ before contact, and $K_e \gg K$ after contact (very stiff object). The initial position of the manipulator is the nominal position. Measurement noise f_n is added to the force sensor. This is a reasonable model for all situations except immediately after contact, where a non-linear system behavior has been experimentally observed[65].

Possible measures for the cost are the *delay-time* or *rise-time* of the deviation from the nominal position, if the concern is about the time taken by the movement before contact. From Figure 4.1, and ignoring the force sensor noise, the closed loop transfer function is

$$\frac{\Delta Z(s)}{F_d(s)} = \frac{\frac{1}{J}}{s^2 + \frac{B}{J}s + \frac{K'}{J}} \tag{4.37}$$

where $K' = K + K_e$ and $\Delta Z(s)$, $F_d(s)$ are the Laplace Transforms of the displacement from the nominal trajectory along z and the desired force f_d respectively.

Hence the following expressions are obtained for the natural frequency w_n and the damping ξ:

$$w_n = \sqrt{K'/J}\ ,\ \xi = B/(2\sqrt{K'J}) \tag{4.38}$$

A reasonable approximation for the delay-time (time elapsed while the system response raises from its initial value to 50% of the final value) when $0 < \xi < 1.2$ is given by[31]

$$t_d \simeq \frac{1 + 0.6\xi + 0.15\xi^2}{w_n} = \sqrt{\frac{J}{K'}} + \frac{0.3B}{K'} + \frac{0.0375B^2}{K'\sqrt{K'J}} \tag{4.39}$$

The delay-time depends on the 3 parameters J, K', B. Assuming a fixed J, t_d increases with B for a fixed K' and decreases with K' for a fixed B.

After contact, the main concern is about the time taken by the force sensed at the manipulator tip to settle down to the desired force f_d. If specifications require the force error in the z direction, $|f_z - f_{zd}|$ to be less than some accuracy by the time the object should be grasped, the system *settling-time* together with the force sensor noise will affect the reliability.

One definition of settling-time as the time the response takes to go from its initial value to within 5% of the final value leads to the following approximation:

$$t_s \simeq \frac{3}{\xi w_n} = \frac{6J}{B} \tag{4.40}$$

Hence the settling-time and consequently the reliability do not depend on K'. Again there is a tradeoff between cost and reliability: for some fixed K and J, if B is increased, the cost (identified here with the delay-time) will increase, but settling-time will decrease and the system will have more chances to attain the desired force before timeout, thus increasing its reliability.

The following instantiations of the definitions above for this particular example summarize and clarify the application of the formalism:

- problem element $\underline{f} = (f_z\ f_{zd})$
- problem solution $S(\underline{f}) = f_{zd}$

- solution approximation $U(\underline{f}, \phi) = f_z + f_n$, as obtained by algorithm ϕ
- algorithm $\phi = \Delta z$, from the position accommodation controller.

4.4.3 Image Processing

The use of stereo vision algorithms to determine the pose (3D position + orientation) of an object in a workspace is usually prone to errors due to

- camera calibration process
- spot noise superimposed on pixel brightness
- pixel resolution

Assuming that camera calibration is reliable enough, the pose estimate degrades with the distance of the object from the cameras due to pixel truncation[6]. From the point of view of a passive vision algorithm, this uncertainty is irreducible. An alternative approach is to use an active vision algorithm to translate a locate object event. An algorithm of this type is described by McInroy and Saridis(1994)[49]. They use N_v different viewpoints to estimate the pose of an object by stereo vision, and reduce the reliability of pose computation by averaging the N_v estimates. One natural measure of cost here would be the number N_v of estimates necessary to increase the reliability beyond some desired level. Notice that an active vision algorithm has additional sources of uncertainty:

- incomplete modeling of manipulator dynamics
- joint position and velocity measurement noise

Still assuming that camera calibration is reliable enough and if pixel noise is uncorrelated from frame to frame, spot noise may be filtered by averaging the pixel brightness from several pictures of the same static scene, taken at different time instants. Suppose now an object is to be located in the image, and a lower bound for the reliability of determining its pose with a given accuracy (with respect to the actual pose) is given. If the computational cost of processing the image is not considered, one possible cost measure would be the number of frames one has to average to obtain the desired reliability.

In the sequel, the cost/reliability analysis of a 2D object location problem using two alternative algorithms is detailed. The images brightness is corrupted by superimposed spot noise only.

Given a rectangle inside a M x M pixels image, the problem is to estimate the position of the rectangle in the image (see Figure 4.2), that is, its central pixel of coordinates (x_c, y_c). The pixels inside the rectangle were initially set to 1, while the outside pixels were set to 0. To simulate spot noise, zero mean gaussian noise was added to the initial value of each pixel in the whole image.

Several assumptions were made with the goal of simplifying the mathematical analysis and the simulation:

- The area of the rectangle is known and equal to $A = (y_e - y_b)(x_e - x_b)$;
- The whole rectangle is inside the boundaries of the image;
- No other objects are present in the image;
- Errors resulting from pixel truncation or computational roundoff were not considered.

To solve the problem two algorithms are proposed:

- The **open loop algorithm** determines the *center of gravity* of the total image, using the equations

$$\begin{aligned}\hat{x_c} &= \frac{\sum_{i=1}^{M}\sum_{j=1}^{M} jb(i,j)}{A} \\ \hat{y_c} &= \frac{\sum_{i=1}^{M}\sum_{j=1}^{M} ib(i,j)}{A}\end{aligned} \tag{4.41}$$

where $b(i,j)$ is the brightness of pixel (i,j).

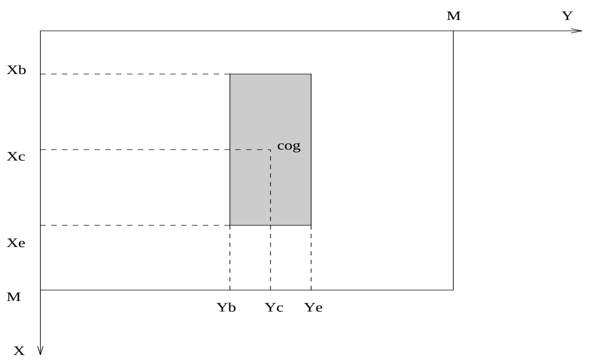

Figure 4.2: Image processed by the two algorithms

- The closed loop algorithm correlates the image (*feedback*) with a pattern rectangle (*reference*) equal in size to the original noise-free rectangle of the image and with the same orientation. The rectangle is assumed to be centered inside the P x P pixels pattern image. The coordinates $(\hat{x}_c, \hat{y}_c)$ of the pixel with the greatest correlation coefficient are the estimates of the rectangle position in the image. Were this a stereo vision problem, and the rectangle in one of the images might be used as the pattern to correlate with the other image.

Both algorithms manipulate images resulting from the average of several image frames to reduce noise.

4.4.3.1 Problem formulation

F is the set of M x M images containing rectangles with size $(x_e - x_b)$ by $(y_e - y_b)$. This includes images resulting of averaging several image frames.

The goal is to compute an ϵ-approximation for $S : F \rightarrow \mathcal{R}^2$, that is to determine an estimate $U(f, \phi)$ of $S(f)$, obtained by an algorithm ϕ, such that

$$\|S(f) - U(f, \phi)\| < \epsilon$$

where $S(f) = (x_c\ y_c)^T$ and $U(f, \phi) = (\hat{x}_c\ \hat{y}_c)^T$, and $\|.\|$ some norm defined on $\Re^2$.

To simplify the analysis, and since what happens in one of the directions is similar to what happens in the other, the error estimate of x_c will be the only one checked

$$|\hat{x}_c - x_c| < \epsilon$$

where ϵ is the accuracy. $\hat{x}_c$, x_c and ϵ may be expressed in pixels or regular length units.

4.4.3.2 Information

The information operations $L_{ij}(f)$, $L_{ij} : F \rightarrow \Re$ give the results

$$L_{ij}(f) = \begin{cases} 1 + n_{ij} & \text{if pixel}(i,j) \in f \\ n_{ij} & \text{if pixel}(i,j) \notin f \end{cases}$$

and so $\mathcal{I} = [L_{111}(f), \ldots, L_{kij}(f), \ldots, L_{NMM}(f)]$ where k denotes the kth frame, N is the total number of averaged frames, M the number of pixels on each side of the image and n_{ij} is a random variable representing the noise at pixel (i,j), $n_{ij} \sim \mathcal{N}(0, {\sigma_b}^2)$, i.i.d.

4.4.3.3 Model of computation

In this simple approach, the set of algorithms is restricted to those that can solve the problem in polynomial time. This means that the cost of getting information is the main concern. For example, if a mobile robot has to stop and get several frames of a scene in order to speedup posterior computations of its locations, a slower algorithm that requires less stopping time for the robot will be preferred. The algorithm may run while the robot is performing other tasks. Sequential computation is also assumed.

Hence, $\text{cost}(\phi) = \text{cost}(\mathcal{I}, f) = cN_{min}^{ol}$, that is, cost is proportional to the minimum number of averages needed by algorithm ϕ to get the error below ϵ. Notice that here f represents an image resulting from the average of N_{min} image frames.

4.4.3.4 Study of the algorithms

The open loop algorithm estimates the center coordinates of the rectangle using equations (4.41). Since the brightness of each pixel is a gaussian distributed random variable

$$p_{B_{ij}}(b_{ij}) \sim \begin{cases} \mathcal{N}(1,\sigma_b^2) & \text{pixel}(i,j) \in f \\ \mathcal{N}(0,\sigma_b^2) & \text{pixel}(i,j) \notin f \end{cases}$$

and the B_{ij}'s are uncorrelated from pixel to pixel (given that they are independent), it can be deduced that, after N averages of distinct frames of the same image and if independent noise from frame to frame is assumed, $p_{\hat{X}_c}(\hat{x}_c) \sim \mathcal{N}(\mu_{\hat{X}_c}, \sigma^2_{\hat{X}_c})$ with $\mu_{\hat{X}_c} = x_c$ and

$$\sigma^2_{\hat{X}_c} = \frac{M^2(M+1)(2M+1)\sigma_b^2}{6A^2N} \tag{4.42}$$

Now, given an accuracy ϵ and a desired reliability R_d, a N will be determined such that $\Pr\{|\hat{x}_c - x_c| \leq \epsilon\} \geq R_d$

$\eta(R_d)$ in $\Pr\{\frac{|\hat{x}_c - x_c|}{\sigma_{\hat{X}_c}} \leq \eta\} = R_d$ can be obtained from the table of standard normal

Equating $\eta(R_d) = \epsilon/\sigma_{\hat{X}_c}$ and using (4.42):

$$N \geq N_{min} = \frac{\eta^2(R_d)M^2(M+1)(2M+1)\sigma_b^2}{6A^2\epsilon^2} \tag{4.43}$$

The inequality in (4.43) comes from the fact that the reliability must be lower bounded by R_d.

The closed loop algorithm looks for the pixel where the noisy output of the correlator achieves a maximum when the pattern is displaced around the image. Due to noise, there is some probability that the wrong pixel is chosen. In order to make the problem tractable, some assumptions have to be made, such as working in 1D again, and considering errors of 1 pixel displacement at most.

If the correlator input $u_{corr}(x) = r(x) + n(x)$, where $r(x)$ is a rectangle of length $x_e - x_b$ and $n(x)$ is gaussian noise of zero mean and variance σ_b^2/N, and the impulse response of the correlator is $r(x_c - x)$, then $y_{corr}(x) = y_r(x) + y_n(x)$ is the correlator output, where $y_r(x)$ is an isosceles triangle of length $2(x_e - x_b)$, centered at x_c and of height $x_e - x_b$, and $y_n(x)$ is gaussian noise with mean x and variance $\sigma_n^2 = \sigma_b^2(x_e - x_b)/N$.

Now, if errors of 1 pixel at most are assumed (that is, ϵ equals the pixel width or 1 pixel, depending on the units used), the desired reliability comes

$$\begin{aligned} R_d &= \Pr\{|\hat{x}_c - x_c| < \epsilon\} \\ &= \Pr\{(y_{corr}(x_c + \epsilon) - y_{corr}(x_c) < 0) \ \wedge \ (y_{corr}(x_c - \epsilon) - y_{corr}(x_c) < 0)\} \end{aligned}$$

Let one assume that the output noise of the correlator is independent from pixel to pixel. This is not actually true, but it allows us to proceed. Since the two random variables are correlated, under this assumption a smaller probability will be obtained, hence resulting in an upper bound for N_{min}. Then

$$R_d = \Pr\{y_{corr}(x_c + \epsilon) - y_{corr}(x_c) < 0\} \ \Pr\{y_{corr}(x_c - \epsilon) - y_{corr}(x_c) < 0\}$$

Noticing that the sum of two random variables jointly and marginally gaussian is another gaussian distributed random variable[61]:

$$R_d = \overset{2}{\Pr}\{z < 0\}, \ z \sim \mathcal{N}(-\epsilon, 2\sigma_n^2)$$

or

$$\sqrt{R_d} = \Pr\{\frac{z + \epsilon}{\sqrt{2}\sigma_n} < \eta\}$$

where $\eta(R_d)$ can be read from a table of standard normal and is made equal to $\frac{\epsilon}{\sqrt{2}\sigma_n}$, hence the upper bound for the minimum necessary number of averaged frames is

$$N_{min}^{cl} \leq N_{minup} = \frac{2\eta^2(R_d)\sigma_b^2(x_e - x_b)}{\epsilon^2} \tag{4.44}$$

Comparing N_{minup} and N_{min}^{ol} for the open loop algorithm (4.43), it may be noticed that the closed loop upper bound on the number of averages does not depend on the size of the image M, while the minimum number of averages for the open loop algorithm increases with M, thus it is possible, with a reasonable ratio of image size to pattern size, to show that the closed loop algorithm upper bound N_{minup} will be below the actual value of N_{min}^{ol} for the open loop case. Simulations show that (4.44) is a loose upper-bound and that in practice the cost of the closed loop algorithm is much smaller, for the same reliability. If the cost of processing information was also considered, other intermediate solutions between the two algorithms would have less cost, since the open loop algorithm is computationally faster.

4.4.3.5 Simulation results

The open loop algorithm was tested with different sets of parameters as follows:

set 1: $R_d = 90\%$, and $\epsilon = 0.1$ pixel and 0.2.

set 2: $\epsilon = 0.3$ pixel, and $R_d = 90\%$ and 95%.

Each of the setups was tested with standard deviation of pixel noise $\sigma_b = 0.1$ and $\sigma_b = 0.3$. Each side of the image had 32 pixels and the rectangle had 13 pixels in the x direction, 9 in the y direction.

σ_b	0.1		0.3	
ϵ	$C = N_{min}^{ol}$	R	$C = N_{min}^{ol}$	R
0.1	73	0.8733	655	0.8933
0.2	18	0.8333	164	0.8533

Table 4.1: Actual reliability and cost of open loop algorithm for a desired reliability of 90 %.

The simulations were made in PRO-MATLAB Version 3.5i, running on a Sun SparkStation. The results are presented in tables 4.1 and 4.2 for parameter sets 1 and 2, respectively. For each setup the reliability obtained from an average of 150

σ_b	0.1		0.3	
R_d	$C = N^{ol}_{min}$	R	$C = N^{ol}_{min}$	R
90 %	8	0.9267	73	0.8733
95 %	11	0.9400	103	0.9467

Table 4.2: Actual reliability and cost of open loop algorithm for an accuracy $\epsilon = 0.3$ pixel.

runs is shown. The actual number of frames used in each run was slightly greater than the cost, i.e. the minimum number of frames theoretically required. Notice that any N above N^{ol}_{min} improves the reliability, for the same ϵ. However, because the worst case was sought, the simulation was confined to values of N immediately above N^{ol}_{min}.

In general, the outcomes agree quite well with the expected results. In some cases, reliability is slightly lower than expected. This may be explained by the fact that MATLAB's random number generator does not assure complete independence of the output values, hence additional terms would be present in equation (4.43), raising the lower bound on the number of averages.

The closed loop algorithm was simulated under the same setup. By trial and error, one determines the standard deviation of the superimposed noise needed to obtain reliabilities close to those of the open loop case with the same number of averages. Table 4.3 shows these results. ϵ was made equal to 1 pixel, because the closed loop algorithm can not achieve sub-pixel resolution.

For the same accuracy ϵ and number of frames $N^{ol,cl}_{min}$, the open loop algorithm has the same reliability of the closed loop algorithm only for noise standard deviations one order of magnitude smaller.

Both results show that, given a desired accuracy ϵ and different environmental conditions (symbolized by different pixel noise variances), the cost of the object location algorithms increases with increasing demand on the reliability.

$N_{min}^{ol,cl}$	R_{ol}/σ_b	R_{cl}/σ_b
4	98%/0.2	99%/2.0
9	97%/0.3	98%/3.0

Table 4.3: Compared reliability and cost for the open loop (ol) and closed loop(cl) algorithms.

If the number of averages is constrained to some value, the open loop algorithm can only attain the reliability of the closed loop algorithm under a much more favorable environment. Hence, for the same cost and under the same environment, reliability would distinguish the two.

Also, if different algorithms of both types, distinguished by the choice of different N at design time, were available, a combination of the cost and reliability associated to each of them would help in the selection of the most reliable algorithm among those constrained by some cost.

4.5 Summary

In this chapter, the formalism of Information-Based Theory of Complexity helped in obtaining a joint definition of Cost and Reliability for the different algorithms composing a feasible set for a problem. The feasibility is determined by the accuracy desired for the solution of the problem and the radius of information of the problem.

A problem must not be confused with the algorithms capable of its solution. $\epsilon - complexity$ was defined for a problem whose specified accuracy is ϵ, but the focus of this work is on the reliability and cost of the feasible algorithms.

If the problem under consideration requires some level of accuracy, given a desired reliability, the formalism may be used to measure the cost of the alternative algorithms. On the other hand, given some number of operations performed by an algorithm, or the amount of resources it uses, the maximum possible reliability of the algorithm can be determined. Hence, given a specified accuracy for a problem, the reliability and the cost of the algorithms which solve the problem with that accuracy can be determined and combined in a cost function which was also introduced.

The concepts of *information* used here and *structured information* in the sense of Shannon are different. Shannon's information refers to the *information content* or the information *conveyed* by a message, while this work deals with the amount of information *contained* in the message. However, when one seeks the amount of information needed to limit uncertainty to a specified level, as it is the case here, information-based complexity is related to Kolmogorov's notion of ϵ-entropy[82]. ϵ-entropy is in turn related to Shannon's entropy, which measures uncertainty as the information content of a message.

The combined measure of reliability and cost presented in this chapter may be used for the off-line design of IMs, if viewed as an extension of the work by McInroy and Saridis[49], and Musto and Saridis[55]. In the sequel, it will be used to build a cost function which is updated recursively on-line and used to learn the action that minimizes the cost function.

CHAPTER 5
The Intelligent Machine as a Hierarchical Stochastic Automaton

The use of feedback to reduce uncertainty and improve performance is a feature of most existing controllers. Uncertainty increases when higher levels of decision are introduced, as it is the case with a Hierarchical Goal-Oriented Intelligent Machine. In a HGDIM, the uncertainty about which decisions to take is added to the uncertainty about the environment where the machine operates. Furthermore, in a HGDIM performance is related to uncertainty. Reducing the uncertainty improves the performance. Thus, a strategy designed to improve the performance of an IM must use feedback from the environment to reduce uncertainty. However, the feedback related to decision making is different from that related to uncertainty about the environment. An algorithm implementing an action selected by the decision making mechanism interacts with the environment in a precise way, but the decision making structure will only receive an abstract report of success or failure, after the algorithm finishes its job. This characteristic of the response of the environment to more abstract actions, and the need to avoid local minima in the search for optimal decisions, makes LSAs the appropriate solution to model the translation interfaces of a HGDIM, where decisions are taken.

This chapter introduces the modeling by a Hierarchical Learning Stochastic Automaton of the feedback structure and the use of feedback in Hierarchical Goal-Directed Intelligent Machines. This formulation has been proposed in several papers by Lima and Saridis[43, 45, 44]. Section 5.1 states the general assumptions made. In section 5.2 the general definitions of HLSA and environment are instantiated with elements of the closed loop HGDIM-environment. The cost function introduced in chapter 4 and its propagation equations are rewritten under this more general formulation in section 5.3, including entropy propagation equations. There are different types of feedback involved in the HLSA. The hierarchical reinforcement learning scheme, recursively updated by feedback, is explained in section 5.4. The feedback hierarchy is detailed in section 5.5. From an engineering point of view, there is a need for a HGDIM design methodology and the general description of its operation by an execution algorithm. The design methodology and execution algorithm for HGDIMs are presented in section 5.6 and 5.7, respectively. Finally, section 5.8 illustrates the relation between this and past work in the same area.

5.1 Assumptions

Some general assumptions must be made before formalizing the relation between the Hierarchical Learning Stochastic Automaton and the HGDIM, for both practical and theoretical reasons:

- The HGDIM moves inside an environment which can be modeled as a multi-state stochastic process (see Definition 3.1.2).
- The HGDIM can recognize the different states of the model of the environment.
- The environment includes the controlled system and a set of monitoring algorithms. Each time an action of the HGDIM is applied to the controlled system, one or more of these algorithms check a set of environment features to determine if the action was successful or not. Notice that, even though monitoring algorithms are conceptually distinct from the HGDIM execution level algorithms, in the actual implementation the former may be embedded in the latter.
- There is a mechanism of error detection and recovery to detect situations which could compromise the integrity of the HGDIM and/or the environment, and recover from them to resume the learning process. This is an important

assumption, since the learning process is based on task repetition, and most learning comes from failures, even though they are not damaging in most cases.

5.2 The Closed-Loop Intelligent Machine and the Environment

The poor knowledge assumed about the environment discourages the use of *off-line* task selection. The model-based reliability and cost help choosing the best algorithms and tasks to implement the HGDIM, only when the environment conditions are well known. Thus, the strategy chosen for this work consists of assigning a cost and estimating reliability. Furthermore, because the environment responds to actions with failures and successes only, reinforcement learning algorithms must be used to estimate reliability and learn the decision making structure. Finally, since reliability has to be estimated on-line, *random decision methods* for primitive algorithms and tasks selection are necessary, to explore the different alternatives some time before the reliability estimates converge.

The above suggests modeling the translation interfaces of a HGDIM as a HLSA. There are two translation interfaces in a 3-level HGDIM:

- The task translating a command is selected by the **Organization-to-Coordination Translation Interface**;
- The primitive algorithm translating a primitive event is selected by the **Coordination-to-Execution Translation Interface**.

It is thus natural to use a 2-stage HLSA, with one LSA per translation interface. The term *stage* will be used for the HLSA, while *level* refers to the HGDIM.

Given the Definition 3.1.1 of a LSA, the linguistic formulation of task generation in HGDIMs in section 3.3, and the cost function introduced in the previous chapter, the translation interfaces of a Hierarchical Goal-Directed Intelligent Machine are modeled by a Hierarchical Learning Stochastic Automaton as follows:

Definition 5.2.1 *For each member c_i of a set of commands $\{c_i,\ i = 1, \ldots, nc\}$, an HGDIM is defined by a 2-stage generalized HLSA, denoted by the sextuple $\mathcal{IM}_i = (U_i, Y_i, G_i, OC_i, CE_i, C_i)$, where*

- *$U_i = \cup_{k=1}^{ne_i} A_k$ is the set of primitive algorithms capable of translating all primitive events $e_k \in E_i$. E_i is the set of primitive events compatible with command c_i. $ne_i = |E_i|$;*
- *$Y_i = \{0, 1\}$ is the finite input set, with 1 representing a reward and 0 a penalty;*
- *G_i is a stochastic grammar defined by the quintuple $(V_{T_i}, V_{N_i}, \mathcal{R}_i, \mathcal{P}_i, S_i)$, where*
  - *$V_{T_i} = E_i$;*
  - *V_{N_i} is the set of symbols representing events (non-null strings of primitive events $e_i \in E_i$). $|V_{N_i}| = m_i$;*
  - *$\mathcal{R}_i$ is a finite set of productions or rewrite rules in the form*

$$A \rightarrow \alpha \quad or \quad A \rightarrow \alpha B,\ \ \alpha \in V_T^*,\ \ A, B \in V_N$$

  - *$\mathcal{P}_i$ is a finite set of production probabilities assigned by a one to one mapping to the elements of $\mathcal{R}_i$;*
  - *$S_i = c_i$ is the start symbol;*
- *$OC_i = \{\Lambda_1^{oc}, \ldots, \Lambda_{m_i}^{oc}\}$ is a set of Learning Stochastic Automata (one LSA per each subset $\mathcal{R}_{ik},\ k = 1, \ldots, m_i$ of the set of productions $\mathcal{R}_i$ of grammar G_i) corresponding to the top stage or* **Organization-to-Coordination Translation Interface***. LSA $\Lambda_k^{oc} \in OC_i,\ k = 1, \ldots, n_{ei}$ is defined by the 4-tuple $\{Q_k^{oc}, U_k^{oc}, F_k^{oc}, G_k^{oc}\}$, where*
  - *$U_k^{oc} = \mathcal{R}_{ik}$ is the set of productions with the same premise B_k^i corresponding to the kth non-terminal symbol of G_i, or the kth event of the IM for command c_i;*

- $Q_k^{oc} = U_k^{oc}$ *is the finite set of internal states, coincident with the finite set of productions with the same premise* B_k^i. *This further implies that* $G_k^{oc} = I$, *the identity matrix;*
- F_k^{oc} *is a reinforcement scheme such that* $P_k^{oc}(n+1) = F_k^{oc}(P_k^{oc}(n), y_k^{oc}(n))$, *where* $P_k^{oc}(n) = \mathcal{P}_{ik} = \{p_1^{ik}, \ldots, p_{m_{ik}}^{ik}\}$ *is the finite set of production probabilities governing the (random) selection of the production to apply at each step, among the productions in* $\mathcal{R}_{ik}$, *and* $y_k^{oc}(n) \in Y_i$.

Notice that, for reasons related to the propagation of the cost function and its use to reinforce the productions probabilities, the top stage LSA does not have an input set;

- $CE_i = \{\Lambda_1^{ce}, \ldots, \Lambda_{ne_i}^{ce}\}$ *is a set of Learning Stochastic Automata (one LSA per each event* e_i *compatible with command* c_i*) corresponding to the bottom stage or* **Coordination-to-Execution Translation Interface**. *LSA* $\Lambda_k^{ce} \in CE_i$, $k = 1, \ldots, n_{ei}$ *is defined by the quintuple* $\{Y_k^{ce}, Q_k^{ce}, U_k^{ce}, F_k^{ce}, G_k^{ce}\}$, *where*
  - $Y_k^{ce} = Y_i$;
  - $U_k^{ce} = A_k$ *is the set of primitive algorithms capable of translating primitive event* $e_k \in E_i$;
  - $Q_k^{ce} = U_k^{ce}$ *is the finite set of internal states, coincident with the finite set of primitive algorithms capable of translating all primitive events* $e_k \in E_k$. *This further implies that* $G_k^{ce} = I$, *the identity matrix;*
  - F_k^{ce} *is a reinforcement scheme such that* $P_k^{ce}(n+1) = F_k^{ce}(P_k^{ce}(n), y_k^{ce}(n))$, *where* $P_k^{ce}(n) = \{p_1^k, \ldots, p_{n_k}^k\}$ *is the set of primitive algorithms probabilities governing the (random) selection of the primitive algorithm to apply at each step;*
- $\underline{C_i}$ *is a cost vector, whose entries represent the costs of the set of primitive algorithms* $\cup_{k=1}^{ne_i} A_k$ *capable of translating all primitive events* $e_k \in E_i$;

In other words: each command is the starting symbol of a stochastic grammar expressing the constraints imposed by the command to its compatible primitive events. The grammar G_i generates a language T_i, the set of alternative tasks capable of implementing command c_i, $i = 1, \ldots, n_c$. The non-terminal symbols of the grammar correspond to events, which may be viewed as macros of primitive events, to structure the task description. Each subset of grammar productions with the same premise represents one (or more) alternatives on the derivation of the translating task. There is a LSA associated to each of these subsets. The productions in each subset are the actions of the LSA, and their probabilities are learned according to the LSA reinforcement scheme (see section 5.4 below). The set of all these LSAs represents the top stage of the hierarchy. The best task is indirectly learned along time, as a result of learning the best production of each production subset.

At every step, a (sub)optimal task is selected by random decision, based on the current subset of production probabilities. Each task is a string of primitive events, and each of the primitive events may be translated in general by more than one primitive algorithm. To learn the best translations, a LSA is associated to each primitive event. The primitive algorithms for the event are the actions of the event LSA. The set of all these LSAs represents the bottom stage of the hierarchy.

The HLSA modeling the translation interfaces of the HGDIM is depicted in Figure 5.1. To help understanding the conceptual function of the top stage, a command is associated to a LSA whose actions are tasks. This formulation is equivalent to the actual definition of top stage LSA.

Since tasks are possible actions associated with a LSA representing a command, T_i must be a *stochastic language*, hence G_i must be a *proper grammar.*

The following definition of environment controlled by a HGDIM links the HLSA and the cost function defined in the previous chapter.

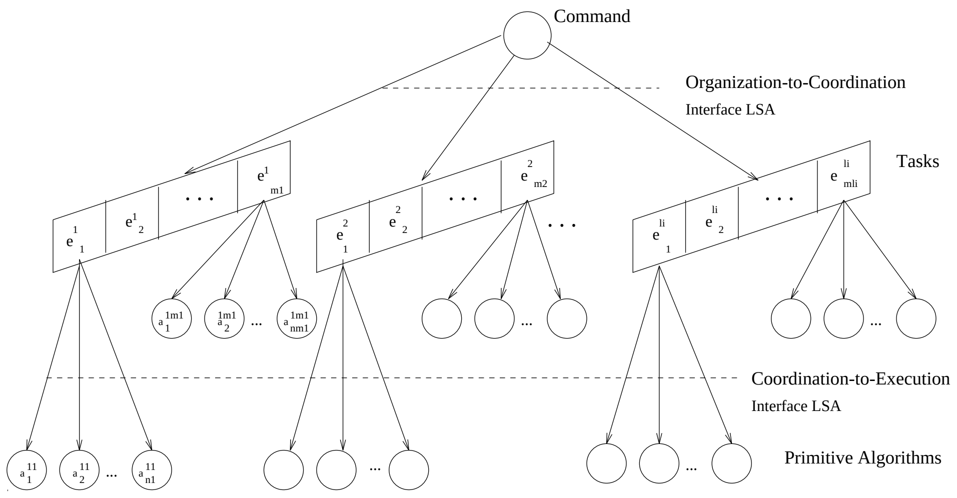

Figure 5.1: HGDIM and Hierarchical Learning Stochastic Automaton.

Definition 5.2.2 *The* **environment** *controlled by a HGDIM is defined by the quintuple* $\{U,\ X,\ Y,\ H,\ R\}$*, where* $U = \{u_1, \ldots, u_s\} = \cup_{k=1}^{ne_i} A_k$ *is the finite input set, or the set of all primitive algorithms (in a total of s) applied by the HGDIM to the environment.* $X = \{x_1, \ldots, x_d\}$ *is a finite set of internal states and* $Y = \{0, 1\}$ *is the output set, where 1 represents a success and 0 a failure.* R *is a matrix with general elements* R_{ij}*, representing the* reliability of algorithm $u_j \in U$, when applied to state x_i *of the environment, according to Definition 4.2.1:*

$$R_{ij} \stackrel{\Delta}{=} \Pr\{u_j \in U \text{ meets specifications}|x_i \in X\} = \Pr\{y = 1,\ y \in Y|u_j,\ x_i\}$$

R determines the rate of failures and successes for each pair (algorithm,state). Finally, $H:\ X^r \rightarrow X$ *is the* state transition function *which generates* $x(n+1) \in X$ *from* $x(n), x(n-1), \ldots, x(n-(r-1)) \in X$.

To simplify notation, the above definition of HGDIM implicitly assumes a single-state stationary environment. Given the general definition of a multi-state environment, the HLSA modeling the feedback activity of the HGDIM must be replicated as many times as the number of states d of the environment. Before a primitive algorithm or a task is reinforced, the state of the environment must be determined to select the appropriate HLSA. More details on the feedback architecture for state determination are given in 5.5.

5.3 Propagation of the Cost Function

In this section, the propagation equations for the cost function originally presented in section 4.3 are rewritten, taking now in consideration the dependence on the state of the environment. Also, the original equations propagated the cost function of tasks. However, the actions of the top stage LSA in Definition 5.2.1 are productions, not tasks. The equations presented in this section propagate the cost of both productions and tasks.

The propagation equations are presented in parallel for the components of the cost function: reliability and cost. At each level of the HGDIM, equation (4.7) applies. All costs are considered normalized to the interval $[0, 1]$.

The **conditional reliability of primitive algorithm** $a \in A_k,\ k = 1, \ldots, n_e$ is defined similarly to the Definition 4.2.1, but it is conditioned by the state x of the environment, and denoted by $R(a|x)$.

The **conditional cost of primitive algorithm** $a \in A_k,\ k = 1, \ldots, n_e$ is defined similarly to the Definition 4.2.2, but it is conditioned by the state x of the environment, and denoted by $C(a|x)$.

These notions can be extended to events, tasks, commands and the overall machine.

Definition 5.3.1 Conditional cost of event $e_k \in E,\ k = 1, \ldots, n_e$ *is the minimum cost among all algorithms translating the event when x is the state of the environment.*

$$C(e_k|x) \triangleq \min_{a \in A_k}\{C(a|x)\} \tag{5.1}$$

Notice the similarity of (5.1) with the definition (4.3) of cost of a problem.

Definition 5.3.2 Conditional reliability of primitive event $e_k \in E$ *is the average reliability among all algorithms translating the event, when* x *is the state of the environment:*

$$R(e_k|x) \triangleq \sum_{j=1}^{n_k} \Pr\{a_j^k|x\} R(a_j^k|x) \tag{5.2}$$

where $\Pr\{a_j^k|x\}$ *is the probability of selecting algorithm* a_j^k *as the action of event* e_k *LSA when* x *is the state of the environment.*

Notice that this definition of primitive event reliability coincides naturally with the definition of average reward of a LSA, given by equation (3.1). This is coherent with the assignment of a LSA to each primitive event.

Definition 5.3.3 Conditional cost of parallel execution of events $e_1, e_2 \in E$ *is*

$$C(e_1//e_2|x) \triangleq \max_{e_1,e_2 \in E} \{C(e_1|x), C(e_2|x)\} \tag{5.3}$$

while the **Conditional cost of** n **events** $e_1, \ldots, e_n \in E$ **executed in series** *is*

$$C(e_1|\ldots|e_n \mid x) \triangleq \frac{1}{n} \sum_{i=1}^{n} C(e_i \mid x) \tag{5.4}$$

The sample mean is used in equation (5.4) to keep the cost normalized.

The successive application of these rules to all primitive events (terminals of the stochastic grammar) composing the consequent of a production (whether they work in parallel or in series) leads to the **conditional cost of a production** r, $C(r|x)$. If there are no primitive events in the consequent, the production cost is zero.

The parallel execution of events is not logically parallel from the reliability point of view. In fact, **all** events must be successful to complete a task, whether they work in parallel or series.

Definition 5.3.4 Conditional reliability of production r *is*

$$R(r|x) \triangleq \prod_{e_k \in \mathcal{E}_r} R(e_k|x) \tag{5.5}$$

The product is defined over the set $\mathcal{E}_r$ *of* primitive events or terminal symbols in the consequent of production r. *If* $\mathcal{E}_r = \emptyset$*, the production reliability is assigned the value 0.5, according to Jaynes' Principle of Maximum Entropy[26].*

The task cost function is obtained from the composition of cost and reliability for all the productions used in the task derivation.

The above definitions of production and task reliability assume that events do not interact, which is hardly true in practice. In fact, failure of one event may imply the failure of another event, or even the success of an event which would be unsuccessful otherwise. For example, if the motion system of a robotic system fails to approach an object with the adequate pose, compliance control will not help the manipulator grasping the object. However, even if the vision system may determine the location of an object with an accuracy out of the error specifications (a *failure*), the motion system may compensate that error with a positioning error.

This suggests that estimating the task (or productions) reliability should alternatively proceed in parallel with the estimation of reliability of events, instead of using the equations to propagate the cost function. The link between the two is implicitly made by the environment. Case study 2 shows an example of separate estimation of productions and events reliability.

The following definitions are not used by the reinforcement scheme, but are presented here for completeness.

Definition 5.3.5 Conditional cost of command c_i, $i = 1, \ldots, n_c$ *is the minimum cost among all tasks translating the command:*

$$C(c_i|x) \triangleq \min_{t \in T_i}\{C(t|x)\} \tag{5.6}$$

Definition 5.3.6 *If the stochastic grammar is unambiguous, that is, if there is only one leftmost derivation for each x, the* **conditional probability of task** t *under state x of the environment is defined by*

$$\Pr\{t|x\} = \prod_{k=1}^{K(t)} \Pr\{r_t(k)|x\} \tag{5.7}$$

where $K(t)$ represents the number of steps in the derivation of t, and $\Pr\{r_t(k)|x\}$ is the probability of the production used in the kth step of the derivation of t.

Definition 5.3.7 Conditional reliability of command c_i, $i = 1, \ldots, n_c$ *is the average conditional reliability among all tasks translating the command:*

$$R(c_i|x) \triangleq \sum_{j=1}^{l_i} \Pr\{t_j^i|x\} R(t_j^i|x) \tag{5.8}$$

The **unconditional reliability** is obtained from the definition of conditional algorithm reliability and from any of the equations (5.2), (5.5) and (5.8) by averaging over all states of the environment:

$$R(*) = \sum_{x \in X} \Pr\{x\} R(*|x)$$

where $*$ stands for algorithm, event, production or command, and X is the set of states of the environment. $\Pr\{x\}$ is the probability of state x of the environment.

The total reliability of the IM is the average reliability over all commands:

$$R(\mathcal{IM}) = \sum_{i=1}^{n_c} \Pr\{c_i\} R(c_i)$$

where $\Pr\{c_i\}$ is the probability of command c_i, $i = 1, \ldots, n_c$ to be issued.

Similarly, the **unconditional cost** is obtained from the definition of conditional algorithm cost and from any of the equations (5.1), (5.3),(5.4),(5.6) and by averaging over all states of the environment:

$$C(*) = \sum_{x \in X} \Pr\{x\} C(*|x)$$

where $*$ stands for algorithm, event, production or command, and X is the set of states of the environment. $\Pr\{x\}$ is the probability of state x of the environment.

The total cost of the IM is the average cost over all commands:

$$C(\mathcal{IM}) = \sum_{i=1}^{n_c} \Pr\{c_i\} C(c_i)$$

where $\Pr\{c_i\}$ is the probability of command c_i, $i = 1, \ldots, n_c$ to be issued.

Entropy was the generic performance measure for a HGDIM originally proposed by Saridis[69]. One of the components of the cost function proposed in this thesis, reliability, can be univocally mapped to entropy at all levels of the HGDIM, as long as its value is greater or equal than 50 %. Otherwise, a reliability of 20 % or 80 %, for example, would correspond to the same entropy.

As for reliabilities, entropies are conditioned by the state x of the environment, that is, $H(*|x)$ is defined based on the conditional reliabilities. Another important measure is the *equivocation*, obtained from $H(*|x)$ as

$$H(*|X) = \sum_{x \in X} H(*|x) \Pr\{x\}$$

and it represents the average information about reliability of $*$ (algorithm, primitive event, production, command), assuming observation of all states of the environment. Unconditional entropies $H(*)$ are similarly derived from the unconditional reliabilities.

Definition 5.3.8 Conditional Entropy of primitive algorithm $a \in A_k$, $k = 1, \ldots, n_e$*:*

$$H(a|x) \triangleq -R(a|x) \log R(a|x) - (1 - R(a|x)) \log(1 - R(a|x)) \tag{5.9}$$

Definition 5.3.9 Entropy of primitive algorithm $a \in A_k$, $k = 1, \ldots, n_e$*:*

$$\begin{aligned} H(a) \triangleq & -\sum_{x \in X} \{R(a|x) \Pr\{x\} \log[R(a|x) \Pr\{x\}] + \\ & (1 - R(a|x)) \Pr\{x\} \log[(1 - R(a|x)) \Pr\{x\}]\} \end{aligned} \tag{5.10}$$

or

$$H(a) = H(a|X) + H(X)$$

where $H(X) = -\sum_{x \in X} \Pr\{x\} \log \Pr\{x\}$ *represents the environment uncertainty, while*

$$H(a|X) = \sum_{x \in X} \Pr\{x\} H(a|x)$$

is the **equivocation of primitive algorithm** $a \in A_k,\ k = 1, \ldots, n_e$ *with respect to the environment, representing the uncertainty about the success of the algorithm.*

Definition 5.3.10 Conditional entropy of primitive event $e_k \in E$*:*

$$\begin{aligned} H(e_k|x) \quad &\triangleq \quad -\sum_{i=1}^{n_k} \Big\{ \Pr\{a_k^i|x\} R(a_k^i|x) \log[\Pr\{a_k^i|x\} R(a_k^i|x)] + \\ &\qquad \Pr\{a_k^i|x\}(1 - R(a_k^i|x)) \log[\Pr\{a_k^i|x\}(1 - R(a_k^i|x))] \Big\} \end{aligned}$$

or

$$H(e_k|x) = < H(a_k|x) > + H(A_k|x) \tag{5.11}$$

where

$$< H(a_k|x) > = \sum_{i=1}^{n_k} \Pr\{a_i^k|x\} H(a_i^k|x)$$

is the average uncertainty about the success of the algorithms translating the event, given the state of the environment, and

$$H(A_k|x) = -\sum_{i=1}^{n_k} \Pr\{a_i^k|x\} \log \Pr\{a_i^k|x\}$$

is the uncertainty about the translation itself.

Definition 5.3.11 Entropy of primitive event $e_k \in E$*:*

$$H(e_k) \triangleq H(A_k|X) + < H(a_k|X) > + H(X) \tag{5.12}$$

When learning is involved, and using the reinforcement scheme of Fu's generalized LSA, the equivocation with respect to the environment about the translation

$(H(A_k|X))$ is reduced along time, due to the convergence to 1 w.p.1 of the action probability of the optimal algorithm.

Similarly, the average uncertainty about the success of the algorithms translating the event ($< H(a_k|X) >$) can be reduced at design time by choosing more reliable algorithms.

Finally, $H(X)$ is the uncertainty about the environment and can not be reduced.

Definition 5.3.12 Conditional entropy of production r*, assuming independent events:*

$$H(r|x) \triangleq \sum_{e_k \in \mathcal{E}_r} H(e_k|x) \tag{5.13}$$

where $\mathcal{E}_r$ is the set of primitive events in the consequent of the production.

Conditional Entropy of a task *is similarly defined. The summation is made over the events composing the task.*

If independence can not be assumed and productions are directly reinforced

$$H(r|x) \triangleq -R(r)\log R(r) - (1 - R(r))\log(1 - R(r)) \tag{5.14}$$

Entropy of production r is defined from the unconditional reliabilities in either case, and so does **Entropy of a task**.

Definition 5.3.13 Entropy of command c_i, $i = 1, \ldots, n_c$ *is*

$$H(c_i) \triangleq H(T_i|X) + < H(t^i|X) > + H(X) \tag{5.15}$$

Definition 5.3.14 Entropy of the HGDIM *is*

$$\begin{aligned} H^{\mathcal{IM}} &\triangleq -\sum_{i=1}^{n_c}\sum_{x \in X} \Pr\{c_i|x\}\Pr\{x\}R(c_i|x)\log[\Pr\{c_i|x\}\Pr\{x\}R(c_i|x)] \\ &= -\sum_{i=1}^{n_c}\sum_{x \in X} \Pr\{c_i|x\}\Pr\{x\}(1 - R(c_i|x))\log[\Pr\{c_i|x\}\Pr\{x\}(1 - R(c_i|x))] \end{aligned}$$

or

$$H(\mathcal{IM}) = H(\mathcal{C}|X) + < H(c|X) > + H(X)$$

where $\Pr\{c_i|x\}$ *is the probability of issuing command* c_i, $i = 1, \ldots, n_c$ *under state* x *of the environment, and* $\mathcal{C}$ *is the set of commands.* $H(\mathcal{C}|X)$ *represents the equivocation with respect to the environment about the command issued.* $< H(c|X) >$ *is the average uncertainty about the success of the command.*

5.4 Hierarchical Reinforcement Learning

The definition of HGDIM left open the selection of the reinforcement or learning scheme. This section defines the learning scheme used at the two stages of the HLSA.

All definitions of reliability in the previous section assume exact knowledge of the conditional reliability of an algorithm. However, as it was explained before, the nature of the environments under consideration requires the estimation of this reliability.

Let $\mathcal{A}_{ij}$ be the probabilistic event corresponding to the successful application of algorithm a_j to state x_i of the environment. Successful application means that the algorithm obtains the desired goals and meets the specifications. Then, $y_{ij} \in Y$, for the set Y in the above definition of environment, is a random variable such that

$$y_{ij} = \begin{cases} 1 & \text{if } \mathcal{A}_{ij} \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

Also from the definition of environment,

$$\Pr\{\mathcal{A}_{ij}\} = \Pr\{y_{ij} = 1\} = R(u_j|x_i) = R_{ij}$$

Furthermore,

$$R_{ij} = E[y_{ij}]$$

Now, if several observations of the random variable y_{ij} are made along time, and since $y_{ij}(1), \ldots, y_{ij}(n_{ij})$ are independent and identically distributed random variables, the following reliability estimate after n_{ij} observations

$$\hat{R}_{ij}(n_{ij}) = \frac{1}{n_{ij}} \sum_{k=1}^{n_{ij}} y_{ij}(k) \tag{5.16}$$

converges w.p.1 to R_{ij}, by the Strong Law of Large Numbers, as $n_{ij} \rightarrow \infty$[64].

The recursive version of the *sample mean estimator* of the reliability (5.16) is

$$\hat{R}_{ij}(n_{ij}+1) = \hat{R}_{ij}(n_{ij}) + \frac{1}{n_{ij}+1}[y_{ij}(n_{ij}+1) - \hat{R}_{ij}(n_{ij})], \quad \hat{R}_{ij}(0) = 0 \tag{5.17}$$

Equation (5.17) is the estimate of performance function (3.5) of Fu's generalized LSA, when $\gamma(n_{ij}+1) = \frac{1}{n_{ij}+1}$ and $\hat{Z}_{ij} = \hat{R}_{ij}$.

Hence, Fu's generalized LSA, described by equations (3.6)- (3.9), can be applied to learn the optimal primitive algorithms translating each of the primitive events at the bottom stage of the HLSA.

In the following definitions of the HLSA reinforcement scheme, the notation $R(u_j|x_i)$ is reduced to a more compact R_{ij}, where x_i is a state of the environment and u_j is one of a primitive algorithm, primitive event or grammar production. Similarly C_{ij} will stand for $C(u_j|x_i)$ and p_{ij} for $\Pr\{u_j|x_i\}$.

Definition 5.4.1 *The reinforcement scheme F_k^{ce} of the* **Coordination-to-Execution Translation Interface** *is defined for primitive event $e_k \in E$, $k = 1, \ldots, n_e$ by the following equations:*

$$\begin{aligned}
\hat{R}_{ij}^{e_k}(n_{ij}+1) &= \hat{R}_{ij}^{e_k}(n_{ij}) + \frac{1}{n_{ij}+1}[y_{ij}(n_{ij}+1) - \hat{R}_{ij}^{e_k}(n_{ij})] && (5.18)\\
\hat{J}_{ij}^{e_k}(n_{ij}) &= 1 - \hat{R}_{ij}^{e_k}(n_{ij}) + \rho C_{ij}^{e_k}(n_{ij}) && (5.19)\\
p_{ij}^{e_k}(n_i+1) &= p_{ij}^{e_k}(n_i) + \frac{1}{n_i+1}[\lambda_{ij}^{e_k}(n_i) - p_{ij}^{e_k}(n_i)] && (5.20)
\end{aligned}$$

$$p_{ij}^{e_k}(0) > 0, \quad \sum_{j=1}^{n_k} p_{ij}^{e_k}(0) = 1, \quad \hat{R}_{ij}^{e_k}(0) = 0 \tag{5.21}$$

$$\lambda_{ij}^{e_k}(n_i) = \begin{cases} 1 & \text{if } \hat{J}_{ij}^{e_k}(n_{ij}) = \min_l\{\hat{J}_{il}^{e_k}(n_{il})\} \\ 0 & \text{if } \hat{J}_{ij}^{e_k}(n_{ij}) \neq \min_l\{\hat{J}_{il}^{e_k}(n_{il})\} \end{cases} \tag{5.22}$$

where $i = 1, \ldots, d$ denotes states of the environment, $j, l = 1, \ldots, n_k$ denote alternative algorithms for e_k, and $n_i = \sum_{j=1}^{n_k} n_{ij}$.

The reinforcement scheme of the top stage of the HLSA is similarly defined, with the important difference that the estimates of the cost function are propagated by the cost and reliability propagation equations of the previous section.

Definition 5.4.2 *The reinforcement scheme F_k^{oc} of the* **Organization-to-Coordination Translation Interface** *is defined for the subset $\mathcal{R}_k \subset \mathcal{R}$ of productions of a stochastic grammar G with the same premise B_k, by the following equations:*

- *Conditional reliability propagation equations (5.2), (5.5), $\forall r \in \mathcal{R}_k$, $\forall x \in X$, with R replaced by its estimate $\hat{R}$;*
- *Conditional cost propagation equations (5.1), (5.3), (5.4), $\forall r \in \mathcal{R}_k$, $x = 1, \ldots, d$.*

In the propagation equations $p_{ir}^{\mathcal{R}_k}$ is now a function of time $p_{ir}^{\mathcal{R}_k}(n)$, updated by the reinforcement scheme:

$$\hat{J}_{ir}^{\mathcal{R}_k}(n_{ir}) = 1 - \hat{R}_{ir}^{\mathcal{R}_k}(n_{ir}) + \rho C_{ir}^{\mathcal{R}_k}(n_{ir}) \tag{5.23}$$

$$p_{ir}^{\mathcal{R}_k}(n_i + 1) = p_{ir}^{\mathcal{R}_k}(n_i) + \frac{1}{n_i + 1}[\lambda_{ir}^{\mathcal{R}_k}(n_i) - p_{ir}^{\mathcal{R}_k}(n_i)] \tag{5.24}$$

$$p_{ir}^{\mathcal{R}_k}(0) > 0, \quad \sum_{j=1}^{n_k} p_{ir}^{\mathcal{R}_k}(0) = 1. \tag{5.25}$$

$$\lambda_{ir}^{\mathcal{R}_k}(n_i) = \begin{cases} 1 & \text{if } \hat{J}_{ir}^{\mathcal{R}_k}(n_{ir}) = \min_q\{\hat{J}_{iq}^{\mathcal{R}_k}(n_{iq})\} \\ 0 & \text{if } \hat{J}_{ir}^{\mathcal{R}_k}(n_{ir}) \neq \min_q\{\hat{J}_{iq}^{\mathcal{R}_k}(n_{iq})\} \end{cases} \tag{5.26}$$

where $i = 1, \ldots, d$ denotes states of the environment, $r, q = 1, \ldots, m_k$ denote alternative productions in $\mathcal{R}_k$ and $n_i = \sum_{j=1}^{m_k} n_{ir}$.

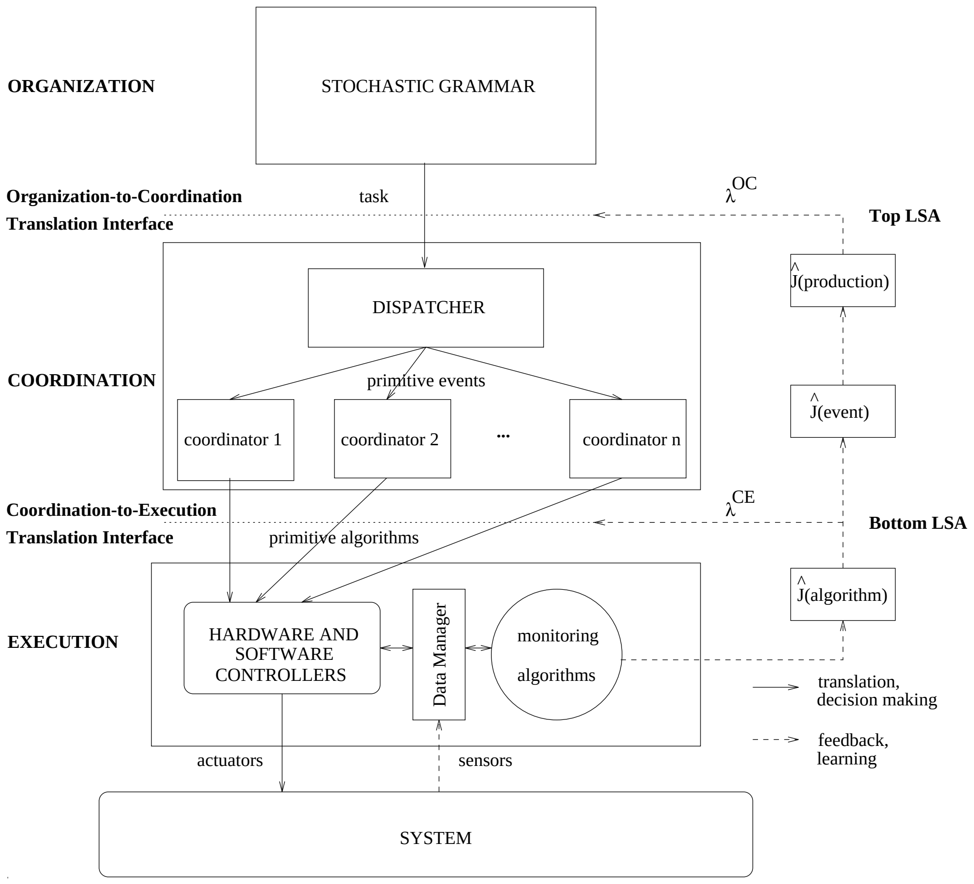

Figure 5.2: Diagram of the HGDIM-Environment loop.

Figure 5.2 shows a block diagram of the closed loop HGDIM-Environment under the proposed definitions of Hierarchical Goal-Directed Intelligent Machine and corresponding reinforcement learning schemes.

Theorem 5.4.1 *Given a command and a state of the environment, the HLSA corresponding to these command and state of the environment (defined by 5.2.1) and its Hierarchical Reinforcement Learning Scheme (defined by 5.4.2 and 5.4.1), are optimal, i. e. the probability of selecting the optimal task and optimal primitive algorithms converges to 1* w.p.1.

The proof of the theorem may be found in Appendix B.

5.5 Feedback Hierarchy

When talking about feedback flow inside the HLSA modeling the translation interfaces of a HGDIM, two types of feedback must be distinguished:

- **Reinforcement feedback**, consisting of *success* (1) and *failure* (0) signals provided by the monitoring algorithms;
- **State of the Environment feedback**, consisting of a set of features characteristic of the current state of the environment, provided by the monitoring and/or other algorithms.

Reinforcement feedback is used by the reinforcement scheme to estimate the cost function and from there the action probabilities.

State of the Environment feedback is necessary to identify the current state of the environment and select the correct HLSA to be updated.

In the previous sections of this chapter, the definitions of reliability, cost and action probability are conditioned by the state of the environment. However, several questions arise regarding the definition of state of the environment at different stages and even within the same stage of the HLSA:

- The HLSA faces an instance of the (well known to Artificial Intelligence researchers) *frame problem*: even though the environment may have several states, a change in the state of the environment may not represent an actual change of the cost function for all LSAs at the bottom stage of the hierarchy. For example in Robotic systems, switching off the lights may be critical for a vision problem, but has no effect on control, path planning or other problems. This may suggest more efficient ways of dealing with the combinatorial explosion of (algorithm, state of the environment) pairs, perhaps by letting each primitive event "decide" from the current features of the environment if, from its point of view, there was a change in the state of the environment.

- The state of the environment effectively "seen" by the top stage of the HLSA is a composition of the states of the environment "seen" by the bottom stage. For example, the state of the environment of a production may be obtained from the current environmental states of the primitive events in the consequent of the production.

The best way to handle these questions is not currently known. However, due to its importance for this work, they were raised here to launch topics of future work in HGDIM. In this thesis, the architecture proposed for the flow of state of the environment feedback through the HGDIM is the following: at the bottom stage of the HLSA, the state of the environment is determined by each primitive event before using reinforcement feedback. The decision about the current state of the environment pertains to each primitive event, and is typically obtained using *Pattern Recognition* techniques. The set of current states of all primitive events is an input feature set for the Pattern Recognition techniques used by the top stage production subsets to determine its own environmental *macro-state.*

Notice that the flow reinforcement feedback is shown as part of Figure 5.2.

5.6 Design Methodology

The following is the proposed design methodology for the IM, based on the discussion of the previous sections:

1. Define the set of commands $\mathcal{C} = c_i,\ i = 1, \ldots, n_c$;

2. Design a stochastic grammar G_i for each of the commands in the command set, representing the alternative tasks for each command. If there is some evidence leading to the assignment of specific initial values of action probabilities to alternative productions of the grammar with the same premise, make them the initial estimates. Otherwise, by Jaynes' Principle of Maximum Entropy, assign

equal initial probabilities to all productions of the same subset $\mathcal{R}_{ik} \subset \mathcal{R}_i$, such that they add up to 1;

3. Define the set E of all primitive events, and its subsets E^i, $i = 1, \ldots, n_c$ for each command, together with the measure of specification error and the specified accuracy ϵ for the problems represented by the events. Notice that specifications are made for the problem represented by each event, **not** for the algorithms translating the event;

4. Assign a set of alternative primitive algorithms to each of the primitive events;

5. Determine the states of model of the environment, from the point of view of every primitive event. Then, determine the macro-states of the model of the environment, from the point of view of top stage production subsets;

6. Assign costs to all (primitive algorithm, state of the environment) pairs. The bottom-up cost propagation may be done at this step, since the cost is not estimated on-line. At the top stage, costs are assigned to (production subsets, macro-state of the environment) pairs;

7. If there is an initial estimate of the reliability (e. g. model based[49, 56]) for any primitive algorithm, use that initial value in the modified reliability estimator (6.10) described in the next chapter. Otherwise assign to every primitive algorithm an initial reliability of 0.5 by Jaynes' Principle of Maximum Entropy. All assignments must be made to (primitive algorithm, state of the environment) pairs;

8. If there is some evidence leading to the assignment of specific action probabilities to alternative algorithms translating an event, make them the initial estimates. Otherwise, again by Jaynes' Principle of Maximum Entropy, assign

equal initial probabilities to all the algorithms translating an event, such that they add up to 1;

9. Design a set of monitoring algorithms whose output are features that can be used by the primitive algorithms to determine its own success or failure. These features can also be used to continuously update a World Model and determine the current state of the environment from the point of view of the primitive algorithm.

The cost of every algorithm translating an event must be determined by the same cost measure. However, in general, different measures may be assigned to different events. For example, if the cost of a vision related event is measured in terms of the number of frames necessary to obtain a reliable estimate, the cost of a motion event in terms of the rise-time of the response, and path planning in terms of the number of knot-points generated, they can still be combined under this definition to obtain the task or production cost. The normalized cost $\rho C \in [0,1]$ must be used not only to balance cost and reliability in the cost function, but also to scale different cost measures among events.

5.7 Execution Algorithm

1. For every primitive event, determine and store the estimate of the initial state of the environment, initial reliability estimates and initial probabilities of selecting primitive algorithms;

2. For every production subset, determine and store the estimate of the initial macro-state of the environment, initial reliability estimates and initial probabilities of selecting productions;

3. Wait for a command $c_i,\ i = 1, \ldots, n_c$;

4. Given the current macro-state of the environment, translate the command by a task selected from the productions of the stochastic grammar for that command by random decision, using the current production probabilities (**Organization-to-Coordination Translation Interface LSA**);

5. For each of the primitive events composing the task, do:

   (a) Given the current state of the environment, select one of the primitive algorithms translating the event by random decision, using the current primitive algorithm probabilities (**Coordination-to-Execution Translation Interface LSA**);

   (b) Execute the primitive algorithm selected in the previous step;

   (c) Check the output of the monitoring algorithm associated to the selected algorithm and update the reliability estimate of the algorithm using (5.18) and the conditional reliability estimate for the primitive event (5.2), given the current state of the environment. From these estimates and the cost, obtain the estimated cost functions (5.19) of the primitive event and its algorithms. Update the probabilities of the primitive algorithms translating the event, using (5.20) and (5.22);

6. Using Equation (5.5) update the reliability of productions having in the consequent the primitive events whose reliability was updated in the previous step. The current macro-state must be taken into account;

7. From the production reliability estimates and the cost propagated by Equations (5.3) and (5.4), build the estimated cost functions (5.23) of the grammar productions. Update the production probabilities for each production subset, using (5.24) and (5.26);

8. For every primitive event, update the estimate of the state of the environment;

9. For every production subset, update the estimate of the macro-state of the environment;
10. **Go to 3**.

5.8 Relationship with Previous Work

This section shows the relationship between the proposed model of the HGDIM and past work in the area by Saridis and his associates.

According to Saridis[70], **Machine Planning** is
"*the ordering of primitive events to form a task.*"

While doing Machine Planning, if the number of primitive events is large, a combinatorial explosion of the number of different possible sequences may happen.

One way of reducing this explosion is to express at design time the natural constraints on events ordering by a grammar. This strategy consists of composing, rather than decomposing, the available information. *A priori knowledge* is expressed by a pre-designed set of primitive events and tasks which are known to be useful in the context where the HGDIM will operate. The machine learns at execution time from experience the best choices among each of these entities.

The stochastic grammar may be directly designed or learned off-line from example tasks[54].

Also according to Saridis[70], **Machine Decision Making** is
"*the process of selecting the sequence (task) with the largest probability of success.*"

The task selection mechanism is implemented in this work by the **Organization-to-Coordination Translation Interface LSA**. For each primitive event composing the task, the **Coordination-to-Execution Translation Interface LSA** implements the primitive algorithm selection mechanism. However, the task and primitive algorithms selected in steady state are not necessarily the most successful (most reliable), but the ones that minimize the cost function (4.7).

Still according to Saridis[70], **Machine Learning** is
"*the feedback process of updating the action probabilities of the tasks and low level algorithms after they are applied.*"

The reinforcement schemes defined by 5.4.1 and 5.4.2 implement this feedback process for primitive algorithms and tasks, respectively.

Previous work on the Analytic Theory of Intelligent Machines proposed solutions for planning, decision making and learning for individual levels of the IM, but not a global solution such as the one introduced in this thesis.

Moed and Saridis[54] proposed a Boltzmann Machine for Planning at the Organization Level. Saridis[70] describes a slightly different stochastic neural network which has more similarities with the stochastic grammar used here. The nodes of the network are stochastic units corresponding to events and primitive events. Each of them has a probability of being active (1) or inactive(0). When one of the units is clamped externally, meaning that the corresponding command was sent to the machine, the sequence of units which become active represents the task chosen to translate the command. The weights associated to the bidirectional arcs connecting pairs of nodes are updated to reflect the probability update of the grammar productions.

Wang and Saridis[92] proposed *Petri Net Transducers* (PNT) for the Coordination Level, described in chapter 2. Beard and Saridis[8] refined this solution recently. However, the main concept remains: each transition of the Hierarchical Petri Net underlying the PNT is translated by a primitive algorithm at the bottom level or by another Petri Net at the upper levels. The bottom level corresponds to the **Coordination-to-Execution Translation Interface LSA** of the HLSA described in this work. Transitions of the PNT Coordinators represent primitive events. The transitions at upper levels correspond to the production subsets of the stochastic grammar or equivalently to non-terminals (events) of the grammar.

These levels correspond to the **Organization-to-Coordination Translation Interface LSA** of the HLSA. The equivalence works if the Dispatcher PNT reads a tape with the events corresponding to the premises of production subsets triggered by a given command. The Hierarchical PNT of the Coordination Level is responsible for the further decomposition of the events into primitive events and from those into algorithms, and also by the reinforcement learning of the translation mechanisms.

5.9 Summary

This chapter introduced the main contribution of this thesis, which is to represent the feedback activity of Saridis' Hierarchical Goal-Directed Intelligent Machine by a Hierarchical Learning Stochastic Automaton. The different types of feedback involved in a HGDIM were also mentioned. A design methodology and a general Execution Algorithm were introduced. The relationship with previous work by Saridis and his associates was analyzed at the end of the chapter.

CHAPTER 6
Convergence Rate and Convergence Acceleration for Stochastic Approximation

The hierarchical reinforcement scheme defined in the previous chapter is based on stochastic approximation methods. The use of stochastic approximation has the advantage of guaranteeing convergence w.p.1 to the optimal action. However, its convergence rate is slow.

In this chapter the transient behavior of the LSA representing the bottom level of the HGDIM is analyzed (section 6.1) to help understanding how long does it take to choose the optimal action most of the times, and what happens while it is not chosen. In section 6.2, two different acceleration schemes documented in the literature on stochastic approximation will be adapted to the formulation of this thesis. Its advantages and disadvantages will be illustrated with some examples and the tradeoffs between using one or more of these methods or having no acceleration at all will be discussed.

In the sequel, the dependence on the state of the environment is not relevant and will be frequently ignored by dropping the corresponding index for simplicity of notation.

6.1 Convergence Rate

Two natural questions concerning the practicality of the proposed hierarchical reinforcement scheme are:

1. Even though the average over a large number of runs converges to the optimal task, is there a risk that some individual run produces a sub-optimal task?

2. How long will it take before the HGDIM selects the optimal tasks and primitive algorithms most of the times?

Before answering these questions, the meaning of *sub-optimal task* or *sub-optimal primitive algorithm* in this formulation must be emphasized: a sub-optimal primitive algorithm (or task) is not necessarily one that may damage the machine or produce very bad performance. According to the design methodology of section 5.6, the primitive algorithms are designed to meet some set of specifications. Due to incomplete modeling and other disturbances, they will not be 100 % reliable. The same can be said about the design of the stochastic grammar for each command. However, a sub-optimal algorithm and/or a sub-optimal task may achieve a reasonable performance each time they are selected and applied. It is implicitly assumed that a HGDIM is built with the experience of knowing the best algorithms, and not randomly selecting algorithms, expecting the machine to "teach" them how to improve their performance. As such, even during the transient period, the behavior of the machine should be acceptable. This methodology aims at providing a measure to help compare different design alternatives and to guarantee the convergence to the optimal operation for a specific design with various alternatives. Given the above, the speed of convergence of the learning process is not as important as in other applications.

The answer to question **1** above is **no**, as long as one waits long enough. To understand this statement, suppose $p(n)$ is a sequence of random variables (in this case representing the evolution of the probability of the best action). The definition of convergence of $p(n)$ to 1 w.p.1 states[61]:

$$\Pr\{\lim_{n\to\infty} p(n) = 1\} = 1$$

or, equivalently

$$\forall \epsilon, \delta > 0 \exists n_0(\delta, \epsilon) \ni \Pr\{\sup_{n>n_0} |p(n) - 1| < \epsilon\} > 1 - \delta$$

That is, it is always possible to find a n_0 such that in **most** of the runs $p(n)$ will be in a neighborhood ϵ of 1 for **every** $n > n_0$. In general, when $\delta \to 0$ and $\epsilon \to 0$, $n_0 \to \infty$. Hence, the optimal action will be chosen for **all** runs when $n \to \infty$.

To help answering the second question, the following Lemma will be proved first:

Lemma 6.1.1 *Consider the reliabilities $R_1, \ldots, R_m$ associated to m algorithms capable of translating some primitive event. With no loss of generality, suppose the algorithms are ordered by increasing values of reliability, i. e. u_m is the most reliable algorithm, u_{m-1} the second most reliable algorithm and so on. Let $\Delta = R_m - R_{m-1}$. Assuming that R_m is unique, $\Delta > 0$. If $\hat{R}_i(n_i)$ denotes the sample mean estimator of R_i after u_i is applied n_i times, the following holds:*

$$\Pr\{\hat{R}_m(n_m) > \hat{R}_j(n_j),\ j = 1, \ldots, m-1\} > \prod_{i=1}^{m}(1 - \frac{4R_i(1-R_i)}{n_i \Delta^2}), \quad \forall i = 1, \ldots, m \tag{6.1}$$

Proof (partially derived in [79]): Let $\mathcal{A}_i$ be the probabilistic event corresponding to the successful application of algorithm u_i to the environment. Then, y_i is a random variable such that

$$y_i = \begin{cases} 1 & \text{if } \mathcal{A}_i \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

And the following equations hold:

$$\begin{aligned} \Pr\{y_i = 1\} &= R_i \\ E[y_i] &= R_i \\ \text{var } y_i &= R_i(1 - R_i) \end{aligned}$$

The sample mean estimator of R_i is given by

$$\hat{R}_i(n_i) = \frac{1}{n_i} \sum_{k=1}^{n_i} y_i(k)$$

with mean and variance

$$\begin{aligned} E[\hat{R}_i(n_i)] &= R_i \\ \text{var } \hat{R}_i(n_i) &= \frac{R_i(1-R_i)}{n_i} \end{aligned}$$

Using Chebyshev's inequality, the following holds for $i = 1, \ldots, m$:

$$\Pr\{|\hat{R}_i(n_i) - R_i| < \Delta/2\} \geq 1 - \frac{4}{\Delta^2}\text{var } \hat{R}_i(n_i) = 1 - \frac{4R_i(1-R_i)}{n_i\Delta^2} \qquad (6.2)$$

This inequality presents a lower bound for the probability that each estimate $\hat{R}_i(n_i)$ is in the interval $[R_i - \Delta/2,\ R_i + \Delta/2]$. By definition of Δ, a lower bound for the probability that the reliability estimate of the optimal action has the greatest value among all reliability estimates is given by

$$\Pr\{\hat{R}_m(n_m) > \hat{R}_j(n_j),\ j = 1, \ldots, m-1\} > \prod_{i=1}^{m} \Pr\{|\hat{R}_i(n_i) - R_i| < \Delta/2\}$$

From this inequality and Equation (6.2), Equation (6.1) holds. $\Box$

A similar reasoning leads to an approximate measure of convergence speed for the reliability estimates: the sample mean estimator converges *in probability*, by the weak law of large numbers[61], for all $i = 1, \ldots, m$:

$$\lim_{k_i \to \infty} \Pr\{\hat{R}_i(k_i) = R_i\} = 1$$

or equivalently

$$\forall \epsilon, \delta > 0 \exists n_{i\min}(\epsilon, \delta) < \infty \ni \ \Pr\{|\hat{R}_i(k_i) - R_i| < \epsilon\} > 1 - \delta, \quad \forall k_i > n_{i\min}$$

For $\epsilon = \Delta/2$, a lower bound is given by Chebyshev's inequality, for each $i = 1, \ldots, m$:

$$\Pr\{|\hat{R}_i(n_{i\min}) - R_i| < \Delta/2\} > 1 - \frac{4R_i(1-R_i)}{n_{i\min}\Delta^2}$$

where $\Delta > 0$ is again the difference between the actual reliabilities for the two best actions.

If δ is given,

$$n_{i_{\min}} = \frac{4R_i(1-R_i)}{\delta\Delta^2}, \quad i = 1, \ldots, m \tag{6.3}$$

and, by Lemma 6.1.1:

$$\Pr\{\hat{R}_m(k_m) > \hat{R}_j(k_j),\ j = 1, \ldots, m-1\} > (1-\delta)^m = P_{LB}\ \ \forall k_i > n_{i_{\min}},\ i = 1, \ldots, m \tag{6.4}$$

Hence, if a lower bound P_{LB} is specified for the probability that the reliability estimate of the optimal action has the greatest value among all reliability estimates, δ can be obtained from Equation (6.4). Given δ, the step $n_{i_{\min}}$, $i = 1, \ldots, m$ after which the lower bound holds is given by Equation (6.3) for each of the estimates of reliability. Notice that $n_{i_{\min}}$ is an upper bound for the actual $n_{i_{\min}}$: the number of steps needed to reach exactly P_{LB} is less than $n_{i_{\min}}$, but no more than $n_{i_{\min}}$ steps are needed to guarantee (6.4).

According to the reinforcement scheme (3.6-3.9), when the number of algorithms capable of implementing a primitive event is m, the knowledge of $n_{i_{\min}}$, $i = 1, \ldots, m$ gives approximate information about the step after which the probability of the optimal action being rewarded is lower bounded by P_{LB}. In fact, if $n_{\min} = \sum_{i=1}^{m} n_{i_{\min}}$, *the probability of the optimal action being rewarded after the* $n_{\min}$*th probability update is lower bounded by* $(1-\delta)^m$. Suppose $P_{LB} = 95\%$. This means that in 'more than 95' out of 100 runs, the estimate of the most reliable action will in fact be larger than the other estimates. Hence, for these 'more than 95 runs', the optimal action will be rewarded by increasing its probability. For high P_{LB}s, the optimal action will become the most probable in a few steps after $n_{\min}$.

This will be now illustrated with an example concerning a 2-actions LSA, for which $m = 2$ in equation (6.3) above. Given a desired lower bound P_{LB}, $n_{\min} = n_{1_{\min}} + n_{2_{\min}}$ depends on the reliabilities R_1 and R_2 of the two actions and on the difference between them, $\Delta = R_2 - R_1$ (assume $R_2 > R_1$).

R_2	0.70	0.75	0.80	0.85	0.90	0.95	1.00
$n_{\min}$	897	848	780	692	585	458	312

Table 6.1: $n_{\min}$ for a 2-actions LSA. $\Delta = 0.2$ and P_{LB} = 0.9

Δ	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
$n_{\min}$	10838	2885	1343	780	508	355	259	195	149	115

Table 6.2: $n_{\min}$ for a 2-actions LSA. $R_2 = 0.8$ and P_{LB} = 0.9

P_{LB} was fixed at an arbitrary value of 0.9. For this value, Table 6.1 shows the values $n_{\min}$ for a 2-actions LSA, when R_2 increases from 0.7 to 1.0 in steps of 0.05, given $\Delta = 0.2$. Table 6.2 shows the values $n_{\min}$ for a 2-actions LSA, when Δ increases from 0.05 to 0.5 in steps of 0.5, given $R_2 = 0.8$. There are infinite possible combinations of P_{LB}, R_2 and Δ. The values shown in the example sweep a range of typical values (see case studies). For other combinations, $n_{\min}$ is determined from Equations (6.3) and (6.4) above.

The results show that $n_{\min}$ decreases when R_2 and Δ increase, i. e. the larger the maximum reliability is, and/or the larger the difference between reliabilities is, the faster the LSA converges to the optimal decision. The influence of Δ dominates. Even though large $n_{\min}$ correspond to small Δ, it is important to keep in mind that $n_{\min}$ is a loose upper-bound and the lower Δ is, the smaller is the error due to selecting the wrong action, as discussed above.

Lemma 6.1.1 will now be used to investigate the time evolution of the mean value of action probabilities. The probability update equations are rewritten here for the case of a single-state environment:

$$p_i(n+1) \quad = \quad p_i(n) + \frac{1}{n+1}(\lambda_i(n) - p_i(n)), \quad i = 1, \ldots, m \tag{6.5}$$

$$\lambda_i(n) = \begin{cases} 1 & \text{if } \hat{J}_i(n_i) = \min_k\{\hat{J}_k(n_k)\} \\ 0 & \text{if } \hat{J}_i(n_i) \neq \min_k\{\hat{J}_k(n_k)\} \end{cases}, \quad n = \sum_{i=1}^{m} n_i$$

Notice $\hat{J}_i(n_i)$ shares the above properties of $\hat{R}_i(n_i)$. Also, minimizing $\hat{J}$ corresponds to maximizing $\hat{R}$ if the cost is zero, as will be assumed in the sequel. This does not imply any loss of generality, because the cost may be interpreted as a bias added to the complement of reliability.

Taking mean values of (6.5)

$$E[p_i(n+1)] = E[p_i(n)] + \frac{1}{n+1}\{E[\lambda_i(n)] - E[p_i(n+1)]\}$$

and rearranging terms

$$E[p_i(n+1)] = \frac{n}{n+1}E[p_i(n)] + \frac{1}{n+1}E[\lambda_i(n)]$$

Using the general solution of the difference equation describing a time-varying discrete system[9], one obtains

$$E[p_i(n+1)] = (\prod_{k=1}^{n} \frac{k}{k+1})E[p_i(1)] + \sum_{j=2}^{n+1}(\prod_{k=j}^{n} \frac{k}{k+1})\frac{1}{j}E[\lambda_i(j-1)], \quad n \geq 1$$

where $\prod_{k=n+1}^{n} \frac{k}{k+1} = 1$ is assumed.

Due to its particular structure, the solution can be simplified to

$$E[p_i(n+1)] = \frac{1}{n+1}\{E[p_i(1)] + \sum_{j=1}^{n} E[\lambda_i(j)]\} \tag{6.6}$$

Now notice that, assuming all costs equal to zero to simplify the derivation:

$$E[\lambda_i(n)] = \Pr\{\hat{R}_i(n_i) = \max_{k=1,\ldots,m}\{\hat{R}_k(n_k)\}\}$$

If $i = m$, that is, focusing on the probability of the optimal action, and using Lemma 6.1.1:

$$\begin{aligned} E[\lambda_m(n)] &= \Pr\{\hat{R}_m(n_m) > \hat{R}_j(n_j),\ j = 1,\ldots,m-1\} \\ &> \prod_{i=1}^{m}(1 - \frac{4R_i(1-R_i)}{n_i\Delta^2}) \end{aligned} \tag{6.7}$$

where $n = \sum_{j=1}^{m} n_j$.

Hence, making (with no loss of generality) $n_i = n/m, \quad \forall i = 1, \ldots, m$,

$$E[\lambda_m(n)] > \prod_{i=1}^{m}(1 - \frac{4mR_i(1-R_i)}{n\Delta^2}) \tag{6.8}$$

and

$$E[p_m(n+1)] > \frac{1}{n+1}\{E[p_m(1)] + \sum_{j=1}^{n}\prod_{i=1}^{m}(1 - \frac{4mR_i(1-R_i)}{j\Delta^2})\} \tag{6.9}$$

Inequality (6.9) gives a *lower bound of the actual time evolution of* $E[p_m(n)]$. It is a loose lower bound, since the lower bound for $E[\lambda_m(n)]$ is also loose. However, it gives a reasonable idea of what one may expect in the worst case.

Again, a LSA with 2 actions will be used to illustrate these results. Figure 6.1 shows plots of the lower bound of $E[p_2(n)]$ when R_2 is incremented from 0.80 to 1.0 in steps of 0.05. Figure 6.2 shows plots of the lower bound of $E[p_2(n)]$ when Δ is incremented from 0.05 to 0.3 in steps of 0.05. Again, the plots represent a small subset of the possible combinations, chosen due to its representativeness. $E[\lambda_2(n)]$ was made 0 when Equation (6.8) gave a negative lower-bound. Initial probabilities $p_1(1)$, $p_2(1)$ were made equal to 1/2, by Jaynes' Principle of Maximum Entropy.

6.2 Methods of Convergence Acceleration

6.2.1 Fu's Acceleration Scheme

One common measure of the convergence rate of a Stochastic Approximation algorithm is the *mean square error* $E[(\hat{\theta}(n) - \theta)^2]$ between the estimate $\hat{\theta}(n)$ at step n and the actual value θ.

An acceleration algorithm proposed by Fu and his associates[22] consists of replacing $y_{ij}(n_{ij}+1)$ in Equation (5.17) by

$$\frac{1}{n_{ij}+1}\sum_{k=1}^{n_{ij}+1} y_{ij}(k)$$

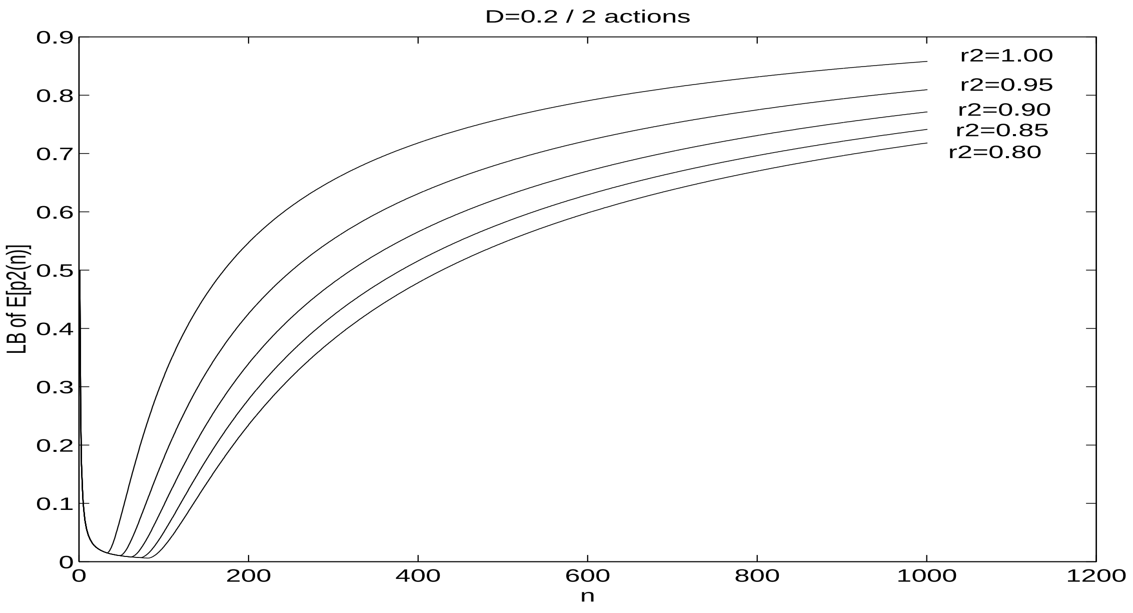

Figure 6.1: Lower bound of $E[p_2(n)]$, for different R_2 when $\Delta = D = 0.2$.

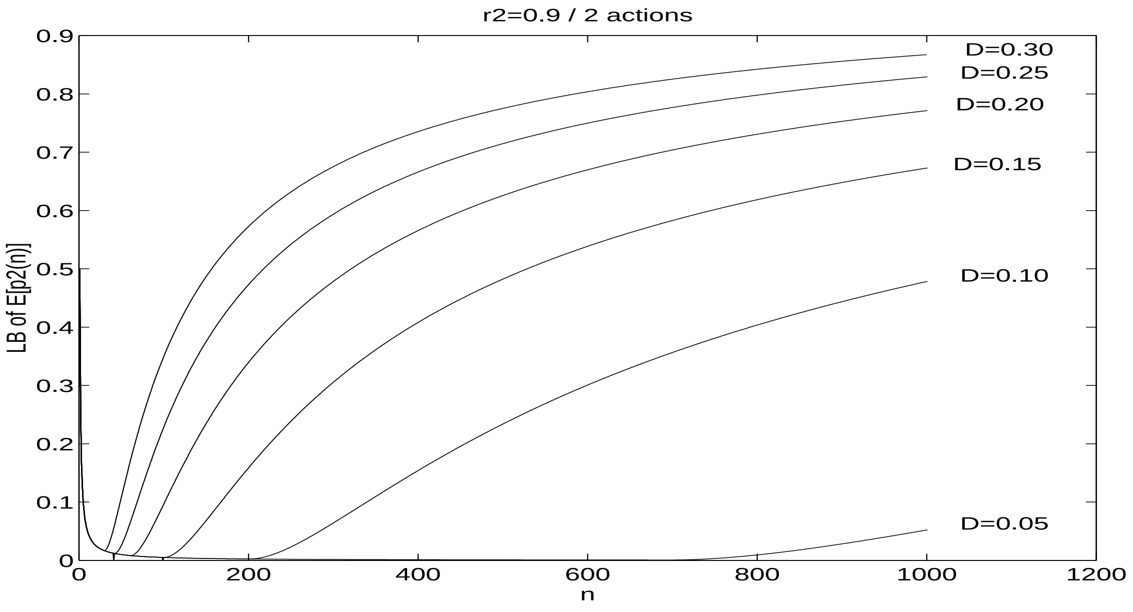

Figure 6.2: Lower bound of $E[p_2(n)]$, for different $\Delta = D$ when $R_2 = 0.9$.

This modified algorithm converges w.p.1 and has a mean square error lower than the original algorithm[22]. Hence, convergence is accelerated and the evolution of the estimates is smoother.

6.2.2 Use of Initial Reliability Estimates

If there is an *apriori* estimate of the sought quantity, the convergence rate may be significantly improved by using that estimate. However, the recursive version of the sample mean estimator of the reliability produces the same estimates as the non-recursive version only when $\hat{R}(0) = 0$ (see Equation (5.17)).

An estimator which includes the *apriori* reliability estimate $\hat{R}(0)$ and a *confidence factor* W in that estimate is given by[20]

$$\hat{R}^W(n) = \frac{1}{n+W}[\sum_{k=1}^{n} y(k) + W\hat{R}(0)]$$

Its recursive version follows:

$$\hat{R}^W(n+1) = \hat{R}^W(n) + \frac{1}{n+1+W}[y(n+1) - \hat{R}^W(n)] \tag{6.10}$$

The estimator is *asymptotically unbiased*

$$\lim_{n\to\infty}\{E[\hat{R}^W(n)] = \frac{n}{n+W}R + \frac{W}{n+W}\hat{R}(0)\} = R$$

and *consistent*

$$\lim_{n\to\infty}\{\text{var } \tilde{R}^W(n) = \frac{nR(1-R) + W^2(R - \hat{R}(0))^2}{(n+W)^2}\} = 0$$

where $\tilde{R}^W(n) = \hat{R}^W(n) - R$, the estimation error.

Furthermore, from the variance of the estimation error, it is possible to analyze the convergence rate for different W. Noticing that

$$\frac{n}{(n+W)^2} = \frac{1}{n} - \frac{2W + W^2/n}{(n+W)^2}$$

the error variance can be rewritten as

$$\begin{aligned}\text{var } \tilde{R}^W(n) &= \frac{R(1-R)}{n} - \frac{2W + W^2/n}{(n+W)^2} R(1-R) + \frac{W^2}{(n+W)^2}(R - \hat{R}(0))^2 \\ &= \text{var } \tilde{R}(n) - \frac{2W + W^2/n}{(n+W)^2}\sigma^2 + \frac{W^2}{(n+W)^2}\tilde{R}^2(0)\end{aligned}$$

where $\sigma^2 = R(1-R) > 0$, $\tilde{R}^2(0) = (R - \hat{R}(0))^2 > 0$ and var $\tilde{R}(n)$ is the error variance for the sample mean estimator. Defining the polynomial in $W \geq 0$

$$P(W) = (\tilde{R}^2(0)n - \sigma^2)W^2 - 2n\sigma^2 W$$

the error variance can be further rewritten as

$$\text{var } \tilde{R}^W(n) = \text{var } \tilde{R}(n) + \frac{P(W)}{n(n+W)^2}$$

The behavior of the convergence rate can be studied by investigating the behavior of $P(W)$. For example, in the trivial case where the initial estimate $\hat{R}(0)$ equals the actual value R, $\tilde{R}^2(0) = 0$ and $P(W) < 0$, $\forall W > 0\ \forall n > 0$. Hence, the estimator $\hat{R}^W(n)$ accelerates convergence in this case, because var $\tilde{R}^W(n) <$ var $\tilde{R}(n)$.

When $\hat{R}(0) \neq R$ different situations exist, depending on the value of W. $P(W)$ has two roots:

$$\begin{aligned} w_1 &= 0 \\ w_2 &= \frac{2n\sigma^2}{\tilde{R}^2(0)n - \sigma^2}\end{aligned}$$

Assuming $n > \frac{\sigma^2}{\tilde{R}^2(0)}$, so that $w_2 > 0$, two situations must be considered. Given R (which determines the value of σ^2) and $\hat{R}(0)$ (which determines the value of $\tilde{R}^2(0)$),

- if the confidence factor $W > w_2$, $P(W) > 0$ and the convergence rate is slower than when the sample mean estimator is used;
- if the confidence factor $w_1 < W < w_2$, $P(W) < 0$ and the convergence rate is faster than when the sample mean estimator is used;

This acceleration method works well when $\tilde{R}^2(0) \simeq 0$, as expected, because the range of W for which $P(W) < 0$ increases significantly, meaning that one can put more confidence in the initial estimate.

To avoid the dependence of w_2 on n, one may prefer to use

$$w_{2\infty} = \lim_{n \to \infty} w_2 = \frac{2\sigma^2}{\tilde{R}^2(0)}$$

as the second root of $P(W)$ and study the behavior of the polynomial as above, but now the results will be valid only when $n \to \infty$.

The stochastic approximation algorithm using this acceleration method does also converge w.p.1. This is proved in Appendix B as part of the proof of Theorem 5.4.1.

6.2.3 Comparison of Acceleration Methods

All combinations of the acceleration methods discussed above were simulated for a 4-algorithms LSA and compared with the case where no acceleration was used. The reliabilities simulated for the 4 algorithms were: $R_1 = 0.8$, $R_2 = 0.85$, $R_3 = 0.9$, $R_4 = 0.95$. These values were chosen to illustrate a situation where convergence is slowed by the reasonably large number of algorithms and the small differences in reliability between them. All plots show the average of 50 runs, with 400 iterations each. Whenever used, initial reliability estimates were $\hat{R}_1(0) = 0.9$, $\hat{R}_2(0) = 0.7$, $\hat{R}_3(0) = 0.75$, $\hat{R}_4(0) = 0.9$. The confidence factors used were $W_1 = 2$, $W_2 = 0$, $W_3 = 0$, $W_4 = 20$, meaning that some confidence is put on the initial estimates of R_1 and R_4, and that the initial estimates for the other algorithms will not actually be used. Notice that $w_{2\infty} = 32$ for algorithm u_1 and $w_{2\infty} = 38$ for u_4, hence in both cases the convergence rate should improve when $n \to \infty$ (where n corresponds to iterations in the figures), because W_1 and W_4 are both smaller than $w_{2\infty}$.

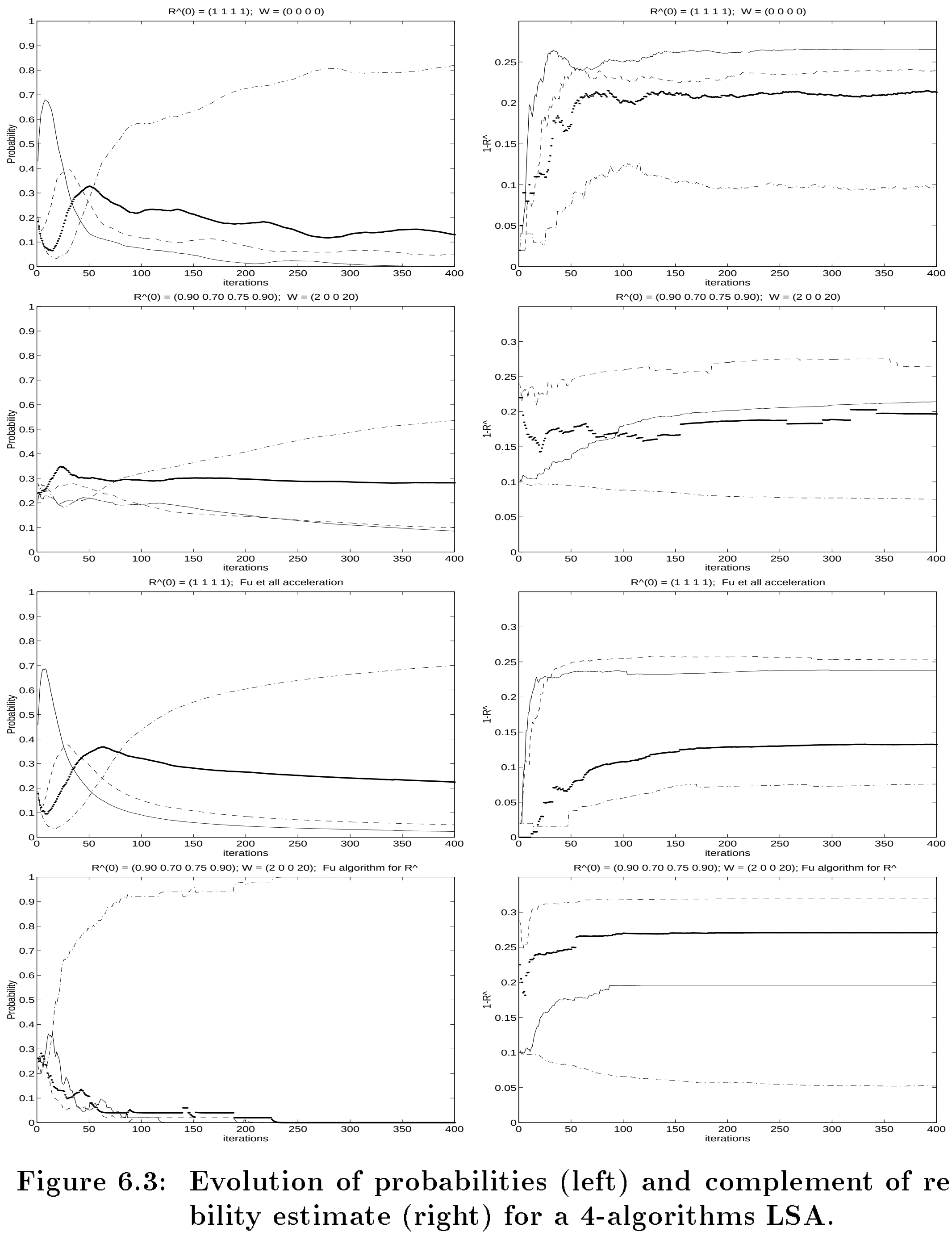

Figure 6.3: Evolution of probabilities (left) and complement of reliability estimate (right) for a 4-algorithms LSA.

The top of Figure 6.3 shows the case where no acceleration was used. The second row of the same figure shows the result of using initial estimates of reliability to accelerate the estimation of the complement of reliability. Even though the convergence is slightly faster for the reliability estimation process, the probability convergence is slower. This illustrates how difficult it is to assign proper initial reliabilities and confidence factors. In the third row, only Fu's acceleration method was used. Both plots are smoother than in the previous examples, but convergence of probabilities is still slow. Finally, the bottom of the figure shows the best among all combinations: when Fu's method and initial reliability estimates are used, convergence of both the complement of reliability estimates and the probabilities gets significantly faster: this is the only case where the optimal algorithm is chosen with probability 1 (for practical purposes) at the end of the 400 iterations.

6.3 Summary

The convergence rate of stochastic approximation algorithms was studied in this chapter. Lower bounds for the evolution of the cost function estimates and the process of learning action probabilities were presented. Two acceleration algorithms were adapted to the problem approached in this work and their performance was investigated both analytically and by simulation.

CHAPTER 7
Case Studies

In this chapter, two case studies in different control areas are presented, to illustrate the application of the theory introduced in the thesis and to demonstrate its flexibility.

Section 7.1 treats the operations management of a glass melting furnace. A small subset of the operations is simulated to introduce in a simple way the main concepts of the theory. Section 7.2 describes a realistic simulation of an intelligent robotic system. This case study is richer in details and number of handled situations.

7.1 Case Study 1 - Operations Management of a Glass Melting Furnace

The operation and control of large scale industrial processes such as water treatment, cement or glass processing is usually a hard task, because it involves several control loops (air, fuel, pressure, temperature) resulting in coupled multivariable systems, with difficult mathematical modeling. The solutions usually implemented are based on conventional controllers (e.g. PID), manually tuned or self-tuned for each control loop. The systems are managed by experienced operators who modify the set points for each loop with the objective of maximizing *efficiency*, constrained by specifications concerning *production levels* and *final product quality*. In recent years, there has been a growing concern about *pollutant production*, and some control strategies include its minimization as another goal.

Process controllers based on expert system shells or fuzzy rules have emerged in the last years[28, 29, 35]. Advantages in terms of efficiency and final product quality have been reported for several practical applications on different process industries. The main reason for these improvements lies in that the *Knowledge Based Systems* used take into account not only overall performance improvement (e.g., maximizing heat transfer to glass), but also the handling of exceptions. For example, if the heat transfer to the glass is maximized, but there is non melted glass in certain areas due to an improper temperature profile, the main goal of the system must temporarily change in order to deal with this problem, even though the goal of maximizing transferred heat is momentarily sacrificed.

These solutions, even though promising, suffer of the usual drawback associated to Artificial Intelligence approaches: the lack of an analytical analysis capable of providing performance measures and a methodology of design. Hence it seems interesting to approach the problem using the methodology described in this work.

7.1.1 Description of the Problem

This section presents the results of modeling a subset of the Operations Management of a Glass Melting Furnace as a HGDIM with the translation interfaces modeled by a HLSA. Due to the current lack of reliable data concerning the performance of the algorithms, the problem is described with detail to illustrate the involved tradeoffs and empirically extrapolate data for the simulations. The modeling issues raised here are typical not only of Glass Manufacturing but also of other industrial processes.

The algorithms referenced in the case study were developed during the EEC ESPRIT-II project AIMBURN ("Advanced Intelligent Multi-Sensor System for Control of Boilers and Furnaces")[2], where they were used under a different approach, described in [40].

A glass furnace usually includes 3 areas[63]:

- Glass Melting Area;
- Working Area;
- Fining Area;

The raw material (batch) is fed to the furnace and is transformed into molten glass in the Glass Melting Area. The homogenization of the glass occurs in the Working Area, and finally its temperature is slowly lowered in the Fining Area, where the glass softly flows to the furnace outlet.

The main objective in the operation of Glass Melting Furnaces is to achieve specified production levels and final glass quality while minimizing fuel consumption. The most important variable that has to be controlled such that specifications can be met, is the *temperature profile* along the furnace. Different temperature profiles lead to different amounts of *heat flux* to the molten glass. Production level and glass quality are indirectly related to the amount and distribution of heat input. Maximization of *thermal efficiency*, defined as the fraction of total energy input (fuel + air energy) transmitted to the glass, corresponds to the minimization of input fuel energy for a desired heat input to the molten glass.

However, direct temperature control by manipulating fuel flow and fuel temperature is not enough, because several factors can disturb the process and hence have to be taken into account. Some examples of such disturbing factors are:

- Inadequate batch composition leading to incorrect glass color. Air/fuel ratio must then be re-tuned for the new conditions, and coloring materials may have to be introduced;

- Changes in molten glass level inside the furnace may introduce impurities in the final glass, or prevent it from reaching the outlet zone of the furnace. Hence the batch input rate must be controlled by the glass level;

- The air pressure inside the furnace lower than the outside pressure allows the leaking of outside cool air having as a result the decreasing of the temperature. Hence a pressure control loop is needed.

- The reduction of pollutant emissions is usually accomplished by decreasing the *excess-air* level (amount of air added to the combustion air specified by the desired fuel flow and air/fuel ratio). However, this decreases the heat flux to the glass and increases the unburnt hydrocarbons leaving the furnace[12].

The management of all these loops and disturbances is usually the responsibility of trained operators who can control the process not only using the information supplied by the flow, the temperature, the pressure, and the glass level meters, but also visual information about flame quality provided by video cameras. Given that information, they may decide to change set-points and air/fuel ratios, or add colorings, to assure regular functioning. Monitoring the location and motion of non melted materials which may be located in, or moving towards, forbidden regions is possible through a video camera momentarily entering the furnace. A barrier of air bubblers is used to prevent the flow of those aggregates of non melted material to reach the furnace outlet. The control of the bubblers air flow can take into account the vision system information. The differential control provided to each furnace burner can also be used to melt the aggregates. The AIMBURN project included a Vision System that provided a set of features describing the non melted aggregates, the flame and the status of burners and bubblers[90].

The furnace considered in the AIMBURN project burns oil and is of the end-fired type. There are 6 burners, installed in the back wall, divided in 2 groups of three, fired alternately each 20 minutes. Each of the 3 burners of a group can be independently controlled. The expertise acquired during the project focused on regular operation and removal of non melted materials, the most important fuel saving and final product quality factors.

One possible design option for these two operations is to consider them as two separate (not independent) goals. Hence, two commands are defined: Regular Operation and Remove Non Melted Materials. Defining the removing of non melted materials as a separate command was the option taken because it is an emergency operation that must be done as soon as it is detected, hence it can not wait for the learning process to reach steady state. However, switching between Regular Operation and Remove Non Melted Materials can be accomplished without human intervention, using the results provided by the vision system: after each task is applied, geometric features of the non melted parts are checked for their proximity of the Fining Area and their dimensions. The features provided by processing the inside-furnace images are the *area* and *mass center* of the whole set of non melted glass parts, a measure of its *eccentricity* in 2-D, and the *radius in the longitudinal direction of the furnace* which measures the extent of the danger of reaching the neighborhood of the bubblers. This information is to be used later by the task translating Remove Non Melted Materials, but at this point the command will be activated only if the materials are too big and too close to the Fining Area.

The simulations described in the next subsection concentrated on the command Regular Operation. The regular operation of glass melting furnaces often faces contentious goals. For example, the goal of maximizing the heat Q_g transferred from flame and radiating walls to the molten glass, conflicts with the need to minimize pollutant levels, expressed by the nitric oxide concentration ($[NO]$) inside the furnace. Carvalho *et al*[12] used a mathematical model to study the effects of combustion excess-air level, air preheating, and fuel composition on the nitric oxide emissions of an industrial glass furnace. The authors concluded that decreasing the excess-air level leads to a decrease of $[NO]$, but also to a decrease of Q_g. Preheating the combustion air increases Q_g, but $[NO]$ is also significantly increased. Furthermore, when the fuel used is a mixture of methane and nitrogen rather than pure methane, both $[NO]$ and Q_g decrease.

To handle these conflicting goals, Farmer and Bryant[16] proposed a control scheme to maximize thermal efficiency, constrained by specifications on the maximum allowed pollutant level. First, they estimated on-line an energy balance model relating the heat flux to the glass Q_g to the total energy input, due to fuel and air flow and temperature. Then, they used this model to predict changes in thermal efficiency, due to changes in fuel and air flow. The optimal controller maximizes efficiency by searching for the optimal fuel and air flow set points, given the current model and constraints on the allowed $[NO]$.

This strategy has its drawbacks:

- experimental data shows[2, 11] that generally thermal efficiency increases when fuel flow decreases. To increase the thermal efficiency, this implies a decrease in Q_g and an increase in minimum residence time of the molten glass inside the furnace, thus leading to a production decrease;

- allowed pollutant levels are fixed at design time and used as constraints, not as part of a global performance index.

An alternative considered here is to have another control scheme to minimize pollutant levels, constrained by the desired production rate, or indirectly by the desired Q_g. Similarly to the other controller, a model of pollutant level formation is needed, to predict changes in $[N0]$ due to changes in excess-air. Such a model is described in [12].

Following the linguistic formulation of task generation in HGDIMs each alternative strategy described above corresponds to separate primitive events. Actually, for reasons made clear below, the two models estimators and the two different optimal controllers have to be separated. The primitive event set E is thus composed by 4 primitive events:

- $e_1 \triangleq$estimate-energy-balance-model;

- $e_2 \triangleq$estimate-pollutant-levels;

- $e_3 \triangleq$max-thermal-efficiency;

- $e_4 \triangleq$min-pollutant-levels;

Even though in the general case each primitive event may be translated by more than one primitive algorithm (or by different parameterizations of the same algorithm), in the sequel only one primitive algorithm per primitive event is considered for simplicity. As an example, the choice of the on-line search technique to determine the optimal fuel and air flow, would distinguish the algorithms translating e_3. In the real furnace all primitive algorithms read information from sensors inside the furnace and modify the set points of PID control loops.

e_1 and e_2 are estimation problems. A natural specification for them is to require the variance of the estimation errors to be less than some accuracy ϵ_1 and ϵ_2, respectively: var $(\hat{Q}_g - Q_g) < \epsilon_1$ and var $([\hat{NO}] - [NO]) < \epsilon_2$. The heat input to the glass is estimated by a_1^1 (the only algorithm translating e_1) from the information provided by the crown, gas and stack temperature sensors. The pollutant levels are estimated by a_1^2 from the current flame characteristics, such as flame length, and oxygen concentration[16]. Both algorithms will be assumed to be 100% reliable.

The specifications for e_3 and e_4 reflect the constraints in each case:

- $|[\hat{NO}] - [NO]_d| < \epsilon_3$. $[NO]_d$ is the desired concentration of nitric oxide;
- $|\hat{Q}_g - Q_{g_d}| < \epsilon_4$. Q_{g_d} is the desired concentration of heat input to the molten glass.

The air-fuel ratio is kept constant. When a_1^3 increases the fuel flow (and the air flow, according to the air-fuel ratio), Q_g increases and so does $[NO]$. Similarly, when a_1^4 decreases the excess-air level, $[NO]$ decreases, and so does Q_g. The two estimators e_1 and e_2 must always be activated prior to the application of any of the optimal controllers. One of them provides the model used by the search algorithm, the other provides information to check the specifications.

a_1^3 and a_1^4 are designed to meet the specifications for the respective events. However, the actual constraints come from indirect indices, such as minimum excess-air, maximum crown temperatures and flame length for e_3, whose actual constraint is $[NO]_d$. Thus, in general these algorithms will not be 100% reliable. The reliability of the algorithms can be checked on-line from the specified Q_{g_d} and $[NO]_d$, and the estimates provided by e_1 and e_2.

The validation of mathematical models of glass furnaces is difficult due to the lack of reliable data. Furthermore, data concerning the performance of optimal controllers is scarce in the case of a_1^3 or non-existent in the case of a_1^4, which is introduced here. The tradeoff between minimizing pollutant emissions and maximizing heat flux to the molten glass is clear from the results of running the mathematical models for different operating conditions[2, 11]. However, the reliability of the estimators and optimal controllers can only be empirically assigned. Similarly, it is natural that computational costs reflect the CPU time spent by the estimators and search algorithms, but this will strongly depend on the actual algorithms, not defined here.

Given these considerations, the case study focus on the convergence of the HLSA to the most reliable decisions, assuming zero costs for all primitive algorithms. Furthermore, a change in fuel composition is simulated in the middle of the experiment. The change is simulated by switching the state of the environment. State 0 corresponds to a pure methane fuel, while in state 1 the fuel is a mixture of methane and nitrogen. Following the results of Carvalho *et al*[12] both $[NO]$ and Q_g are expected to decrease from state 0 to state 1. Hence, it is reasonable to expect that the reliability of a_1^3 will increase, because its constraint on the concentration of nitric oxide will be easier to meet. The reliability of a_1^4 will decrease, because its constraint on the heat flux to the molten glass is harder to satisfy.

Alternative tasks differ by the decision of maximizing thermal efficiency or

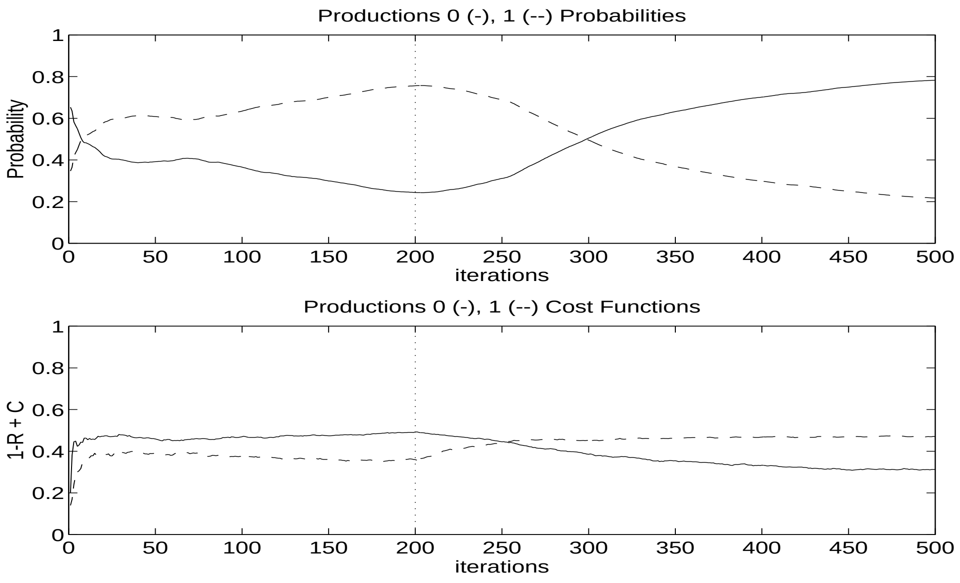

Figure 7.1: Results of case study 1.

minimizing pollutant level. The stochastic regular grammar for the command is:

$$
\begin{array}{lcl}
G & = & (V_T,\ V_N,\ \mathcal{R},\ \mathcal{P},\ \mathrm{S}\ ) \\
V_N & = & \{\ \mathrm{S}\ \} \\
V_T & = & \{e_1, e_2, e_3, e_4\} \\
\mathcal{R} & = & \{ \\
\frac{1}{2} & \mathrm{S} \rightarrow & e_1\ e_2\ e_3 \\
\frac{1}{2} & \mathrm{S} \rightarrow & e_1\ e_2\ e_4\ \}
\end{array}
$$

The numbers to the left of the productions are the initial production probabilities.

7.1.2 Results

Figure 7.1 shows the average over 50 sample functions of the stochastic process corresponding to the evolution of action probabilities (top) and cost functions (bottom) of the 2 productions composing the stochastic grammar for the case study.

event	e_1	e_2	e_3	e_4
R_{0j}	1.00	1.00	0.60	0.70
R_{1j}	1.00	1.00	0.80	0.55
$C_{0j} = C_{1j}$	0	0	0	0

Table 7.1: Reliabilities and costs assigned to the primitive algorithms translating the primitive event set E.

Each sample function consisted of 500 task runs (iterations in the figure). The state of the environment switches from state 0 (pure methane fuel) to state 1 (methane + nitrogen fuel) at iteration 200, but state 0 remains the estimated state for the IM. This was done to test the adaptiveness of the learning scheme.

The rates of successes for the different algorithms were simulated by a Monte Carlo method from the actual reliabilities assigned to each algorithm. No physical model was run. Reliabilities were assigned qualitatively, based on the above discussion. a_1^4 was assumed to be initially more reliable than a_1^3. After state of the environment changes, the reliability of a_1^4 decreases and the reliability of a_1^3 increases, such that it becomes more reliable than a_1^4. Costs and reliabilities are tabulated in Table 7.1.

A recursive sample mean algorithm with *forgetting factor* was used to improve the adaptiveness of cost function and probability estimates. No proof of convergence exists for this algorithm, which resembles the stochastic approximation algorithm described before, but gives more weight to recent samples than to old samples. Its general expression is

$$\hat{R}(n) = \frac{1}{n+W}\sum_{i=1}^{n} \beta^{n-i}(i-(i-1)\beta)y(i) + \frac{W\hat{R}(0)}{n+W} \tag{7.1}$$

where $y \in \{0,1\}$ is the reinforcement signal and the forgetting factor $0 < \beta \leq 1$ is usually made close to 1, so that only the most recent samples are used to compute the mean without being too sensitive to small data changes. W is the *confidence* *factor* on the initial estimate $\hat{R}(0)$ (see chapter 6). No acceleration method was used, and $\hat{R}(0) = 0$, hence the confidence factor was only included in the expression for completeness and future reference.

The recursive version of Equation (7.1) is

$$\hat{R}(n+1) = \beta\hat{R}(n) + \frac{1}{n+1+W}\{[n(1-\beta)+1]y(n+1) - \beta\hat{R}(n)\} + (1-\beta)\frac{W\hat{R}(0)}{n+1+W} \tag{7.2}$$

Notice that Equation (7.2) is the same as Equation (5.17) when $\beta = 1$ and $W = 0$.

The recursive version was used for the estimation of algorithm reliabilities and production probabilities. The forgetting factors used were $\beta = 0.97,\ 0.98$ respectively.

The plots show that in state 0, the HLSA converges to the most reliable task, $e_1e_2e_4$. After the change to state 1, task $e_1e_2e_3$ becomes the most reliable, and the probability of production 0 becomes greater than the probability of production 1. Convergence is slower after the change in the environment, due to the memory of the stochastic approximation algorithms.

7.2 Case Study 2 - An Intelligent Robotic System

The coordination of vision and motion algorithms is one of the typical problems in Intelligent Robotic Systems. This is the subject of the case study described in this section.

7.2.1 Description of the Problem

The workspace setup for Case Study 2 is depicted in figure 7.2.

A manipulator PUMA 560 has to grasp a cylindrical strut whose 3D pose (position + orientation) is roughly known. There is a pair of cameras in the ceiling, overviewing the working space of the manipulator and used by a stereo vision system

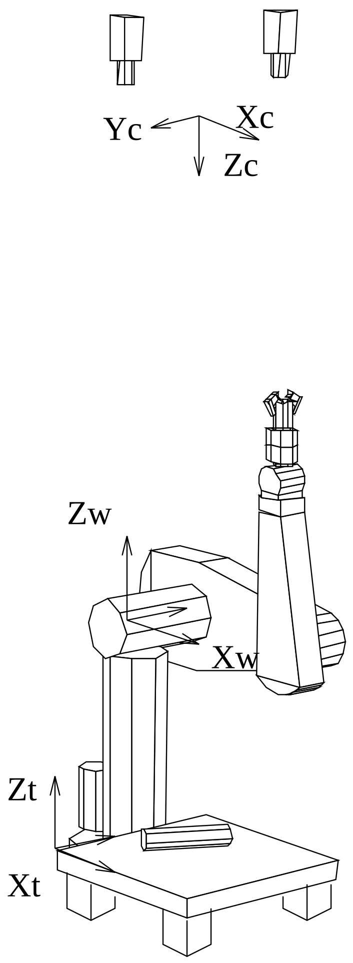

Figure 7.2: Workspace setup for Case Study 2.

to determine more accurately the 3D pose of the object. The manipulator has position, velocity and force sensors. The scene is well illuminated but from time to time lights go off, deteriorating the accuracy of vision algorithms. The environment has 2 states, one corresponding to lights on (state 0) and the other to lights off (state 1).

The only command available is $c =$Grab-Strut.

The event set is composed by 5 events, $E = \{e_1, e_2, e_3, e_4, e_5\}$, where

- $e_1 \overset{\Delta}{=}$move manipulator;

- $e_2 \triangleq$grasp object with compliance;
- $e_3 \triangleq$locate object;
- $e_4 \triangleq$plan trajectory;
- $e_5 \triangleq$grasp object hard;

Event e_1 represents the motion of the manipulator tip along a pre-planned trajectory in joint space. It must not precede e_4. Two Computed Torque algorithms distinguished by their gains (K_p, K_v) (see Appendix A) are capable of implementing e_1, given the desired joints position, velocity and acceleration provided by e_4. The specification for e_1 requires that the quadratic error between the desired and the actual final positions (time t_f) of the manipulator tip in cartesian space, do not exceed the accuracy ϵ_1 specified for the event:

$$(x(t_f) - x_d(t_f))^2 + (y(t_f) - y_d(t_f))^2 + (z(t_f) - z_d(t_f))^2 < \epsilon_1$$

Event e_2 assumes that the tip (tool) of the manipulator is close enough to the object to be grasped and the gripper is opened. Motion driven by some desired force takes the tool to the object. If the pose errors are small, they will be accommodated by the translating algorithms. Then, the force along pre-specified direction(s) will be checked at each step. **If** the sensed force remains within the desired accuracy interval $[f_d - \epsilon_2, f_d + \epsilon_2]$ (where f_d is the desired force along that direction) during a reasonable number of steps, **before** a pre-specified limit number of steps is reached, the gripper will be closed to grasp the object. If either the pose errors are too large when the tool contacts with the strut and/or the desired force is not obtained before the limit number of steps, a failure will be reported.

Even though *Position Accommodation Control* (see section 4.4.2) is used by all algorithms translating e_2 to accomplish compliance control of the manipulator, different parameterizations of the required impedance (similar to a mass, spring and damper system) result in different algorithms with different costs and reliabilities (see Appendix A).

Event e_3 determines the pose of an object using stereo vision algorithms. The uncertainty on pose determination by stereo vision is mainly due to matcher errors when determining which image pixels in the two cameras correspond to the same point in the observed scene. This may be due both to spot noise and pixel resolution[36, 51], and leads to *disparity errors* which affect the 3D pose estimation. The results obtained for the open loop and closed loop algorithms described in section 4.4.3 were used in the simulation of the two algorithms translating e_3 (see Appendix A). In a real-world implementation, each of the algorithms would determine the strut end-points in each of the images to match them and compute the 3D pose of the strut.

Changing the state of the environment deeply affects the two algorithms. Switching the lights off increases spot noise which affects the estimation (by any of the algorithms) of the strut end-points in each of the images , as it was shown in section 4.4.2. This will increase disparity errors and consequently pose estimation errors.

Figuring out a convenient specification for a sensing event such as e_3 is usually hard and application-dependent, because the algorithms do not know any desired set point, in the sense of the specifications for e_1 or e_2. In this case, the measure used was the difference in estimated depth for both ends of the strut. It is assumed that the strut lies in a plane parallel to the camera plane, even though its distance from the latter is not known. Hence, depth should be the same for both extremes. The specification requires the depth difference to be below an accuracy ϵ_3 which equals the depth resolution of the stereo system.

Event e_4 plans a trajectory in joint space whose end-points are the joint-space vectors corresponding to the initial and final pose required for the manipulator

event	e_1		e_2				e_3		e_4	e_5
algorithm	a_1^1	a_2^1	a_1^2	a_2^2	a_3^2	a_4^2	a_1^3	a_2^3	a_1^4	a_1^5
cost	0.55	0.44	0.18	0.14	0.16	0.22	0.35	0.35	0.00	0.3

Table 7.2: Primitive events, primitive algorithms and assigned costs.

tip when moving from a standby position to the grasping position. A *minimum-jerk trajectory generator* introduced by Kyriakopoulos and Saridis[32] is the only algorithm used. It generates the desired joints acceleration, velocity and position at a number of pre-specified points along the trajectory by a method which minimizes the jerk, or third derivative of joint position.

Event e_5 is a non-compliant version of e_2. All compliance will be passive, i. e., a result of the manipulator mechanical compliance. The manipulator is position-controlled only. There is neither a desired force nor any checking of the actual force. Specifications concern the capability of the manipulator tool to comply with errors in pose estimation. This capability is however much smaller than for e_2 (see Appendix A).

A realistic simulation of the algorithms presented here was developed. The details are given in Appendix A.

Table 7.2 shows the costs of the primitive algorithms for the different primitive events. Notice that specifications are made for each event, not for each algorithm. Cost does not change with the state of the environment. Appendix A explains how the costs for the various primitive algorithms were obtained, based on the results of Section 4.4.

During the simulation, the reliabilities of the algorithms were estimated based on the rewards resulting from successes of their application over the environment. The HLSA defined by 5.2.1 and the hierarchical reinforcement scheme (Definitions 5.4.1 and 5.4.2) were used to simulate the HGDIM.

To grab the strut, the system must first estimate the strut pose. Then, it plans a path from the current pose of the manipulator tip to the neighborhood of the strut pose, and moves along that path. Finally, the manipulator slowly approaches the strut and tries to grasp it. Alternative tasks differ by the inclusion or not of compliance and by using or not using stereo vision to refine the *a priori* knowledge of the strut pose. The stochastic regular grammar for the command is:

$$\begin{array}{llll}
G & = & & (V_T,\ V_N,\ \mathcal{R},\ \mathcal{P},\ \mathrm{S}\ ) \\
V_N & = & & \{\ \mathrm{S},\ \mathrm{A},\ \mathrm{B}\ \} \\
V_T & = & & \{e_1, e_2, e_3, e_4, e_5\} \\
\mathcal{R} & = & & \{ \\
0.5 & \mathrm{S} & \rightarrow & e_3\ \mathrm{A} \\
0.5 & \mathrm{S} & \rightarrow & \mathrm{A} \\
1.0 & \mathrm{A} & \rightarrow & e_4\ e_1\ \mathrm{B} \\
0.5 & \mathrm{B} & \rightarrow & e_2 \\
0.5 & \mathrm{B} & \rightarrow & e_5\ \}
\end{array}$$

The numbers to the left of the productions are the initial production probabilities, that is $P(0)$, assigned by Jaynes' Principle of Maximum Entropy.

Productions such as S $\rightarrow$ A, with no terminal symbols in its right-hand side, are assigned zero cost and 0.5 reliability (see section 5.3).

7.2.2 Results

The simulation described in the previous section was run for several different situations in order to point out the main concepts and tradeoffs of the proposed execution and design methodologies, and also to suggest future work.

All results shown are the average over 50 sample functions of some stochastic process, such as the evolution of action probabilities or cost function, except when noted.

event	e_1		e_2				e_3		e_4	e_5
algorithm	a_1^1	a_2^1	a_1^2	a_2^2	a_3^2	a_4^2	a_1^3	a_2^3	a_1^4	a_1^5
$\hat{R}(0)$	0.77	0.90	0.80	0.80	0.50	0.50	0.85	0.93	1.00	0.75
W	10	10	10	10	10	10	50	50	0	20

Table 7.3: Primitive events, primitive algorithms, initial reliability estimates and corresponding confidence factors.

Each sample function consists of 150 task runs (iterations in the figure), except when noted. The production and algorithm probabilities are updated after a task is applied. The initial state of the environment is always state 0 (lights on).

Figures 7.3-7.8 show the evolution of action probabilities and cost functions associated to the main events and productions when the environment is assumed to have only one state – state 0 (lights on). Fu's acceleration technique together with initial reliability estimates were used to obtain the results shown in Figures 7.3, 7.5 and 7.7. The initial reliabilities and confidence factors are tabulated in Table 7.3. The values were picked as approximations of the actual values, from the experience acquired after several simulation runs. No acceleration techniques were used in the other cases. Notice that when the confidence factor $W = 0$, $\hat{R}(0) = 1.00$, because in the actual implementation the complement of reliability is estimated, not the reliability itself.

In all cases the probability of the best action eventually converges to 1.00. However, for events e_1 and e_2 the evolution is slow, due to the close reliability values for the algorithms involved. This initial difficulty of the learning algorithm is enhanced in Figure 7.9, where the decrease of the entropies for the two events is slow when compared with the entropy decrease rate of e_3.

The effect of using acceleration techniques is clear: only the probability of the best algorithm for event e_2 does not reach a close neighborhood of 1.00, but even in this case the acceleration helps a fast distinction between the two best algorithms and the other two.

From the stochastic grammar for the problem and the results shown for the production probabilities, it may be noticed that task $e_3\ e_4\ e_1\ e_2$ is the optimal task. This corresponds to using stereo for pose estimation and compliance for the grasping subtask, as expected. Figure 7.10 shows that task entropy decreases with learning.

Figures 7.11- 7.13 show an example of adaptation of the IM to an unacknowledged change in the state of the environment. In this case each figure shows only one sample of the stochastic processes. The state of the environment switches from state 0 (lights on) to state 1 (lights off) at iteration 150, but state 0 remains the estimated state for the IM. The simulation consisted of 600 task applications. No acceleration method was used. To improve adaptiveness, Equation (7.2) was used for the estimation of algorithm reliabilities and probabilities, and production probabilities. The forgetting factors used were $\beta = 0.97,\ 0.98,\ 0.98$ respectively.

After the state switching, production 0 is no longer the best production, since the performance of the stereo algorithms deteriorates under poor lighting conditions, and the initial rough estimate of the strut pose is preferred. In this run, the IM learned the change.

At a first glance, the state switching should only affect event e_3 and consequently productions 0 and 1. However, figure 7.12 shows that the cost function of event e_2 does also change. Even though this change is not reflected in the evolution of algorithm probabilities (which is messy in this example just because of the close values of the different reliabilities), the increase in the cost function shows the existence of a dependence between events. In the example, the reliability of the grasping events decreases due to the decrease in the quality of the object pose estimates when lights go off. The cost function of event e_5 does not change, only because production 4 is not applied since very early.

An interesting issue here is the comparison between the direct reinforcement of task probability with the indirect reinforcement resulting from the propagation of the cost function. The latter is the method described elsewhere in the thesis, while the former consists of using sensor information (e. g., a cross-fire sensor when the goal is to grasp an object) to directly determine the success or failure of a task and provide the appropriate reinforcement signal to the top level generalized LSA, whose reinforcement scheme will in this case be equal to the bottom level LSA reinforcement scheme (see Definition 5.4.1).

The advantage of the direct reinforcement method is a more realistic estimate of the task cost function, since the propagation equations for the reliability assume independent events, which is not always true, as it was evident in the adaptive example just shown. However, the interaction between the two independent LSAs in the direct reinforcement case has not been studied yet, thus convergence can not be guaranteed. Of course, if there is no way of directly checking the success or failure of a task, only the indirect reinforcement method can be used.

The evolution of production probabilities and cost functions when direct task reinforcement is used is plotted in Figure 7.14. Comparing with the indirect reinforcement method (Figure 7.7), two main differences are noticeable:

- The cost functions steady state values are larger when direct reinforcement is applied. Since the same algorithm costs and cost propagation equations were used, this means that the actual task reliability is smaller than the task reliability obtained by propagating the event reliabilities. The evolution of event estimated reliabilities when direct task reinforcement was used (not shown) is very similar to the indirect reinforcement case, as expected.

- Production 1 probability is initially larger then production 0 probability for the direct reinforcement case, while production 0 has always larger probability in the indirect case. In the latter, cost and reliability of production 1 are initially assigned to 0 and 0.5, remaining equal to these values along the simulation. Using the direct task reinforcement method, the complement of reliability for both productions is learned along time. The initial estimate of the complement of reliability for both productions is equal to zero. The costs are 0.35 for production 0 and 0 for production 1. Hence the initial cost function of production 1 is smaller than that of production 0 in the direct reinforcement case, and larger in the indirect reinforcement case.

The main drawback associated to direct task reinforcement is that there is no proof of its convergence. However, these experiments suggest the method converges and obtains more accurate estimates of production cost function and probabilities.

7.3 Summary

Two case studies were presented where the proposed formalism is applied the Operations Management of a Glass Furnace and an Intelligent Robotic System requiring coordination among vision and motion control. The results show the convergence of the reinforcement schemes at the different levels of the IM hierarchy. Different tradeoffs concerning direct versus indirect task reinforcement, convergence speed and adaptiveness to changes in the state of the environment were also analyzed.

The overall entropy decrease with learning was also shown in Case Study 2.

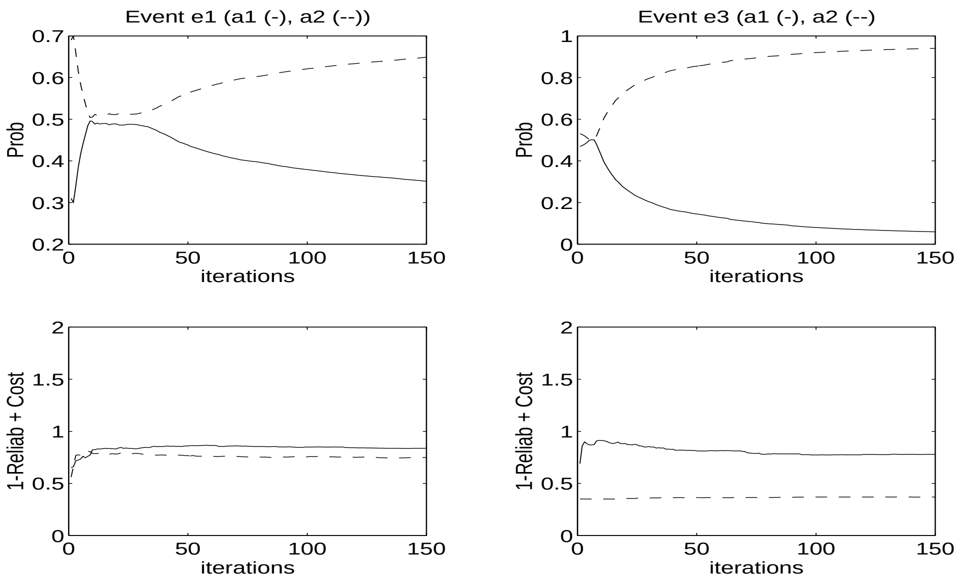

Figure 7.3: Evolution of probabilities and cost functions for algorithms translating events e_1 and e_3.

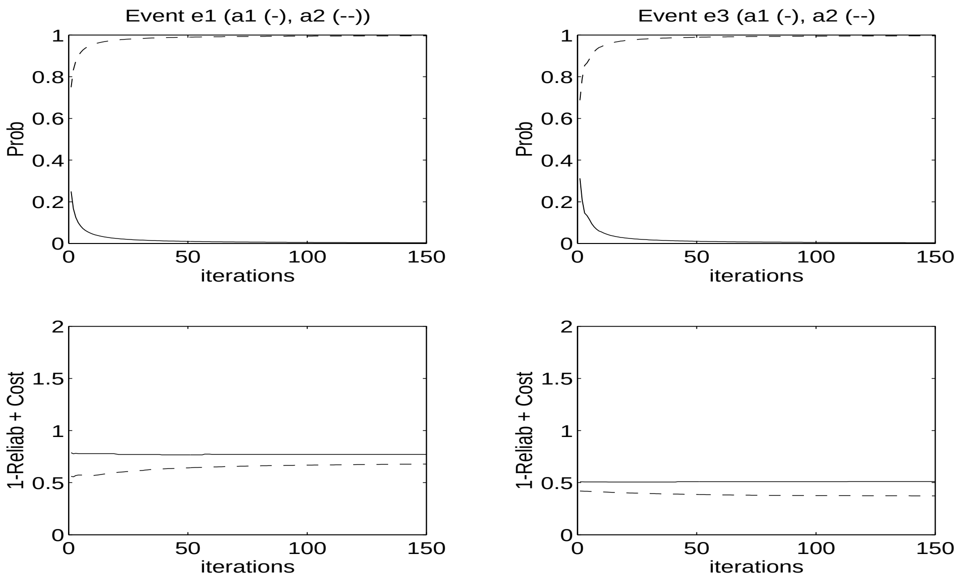

Figure 7.4: Evolution of probabilities and cost functions for algorithms translating events e_1 and e_3, when acceleration method is used.

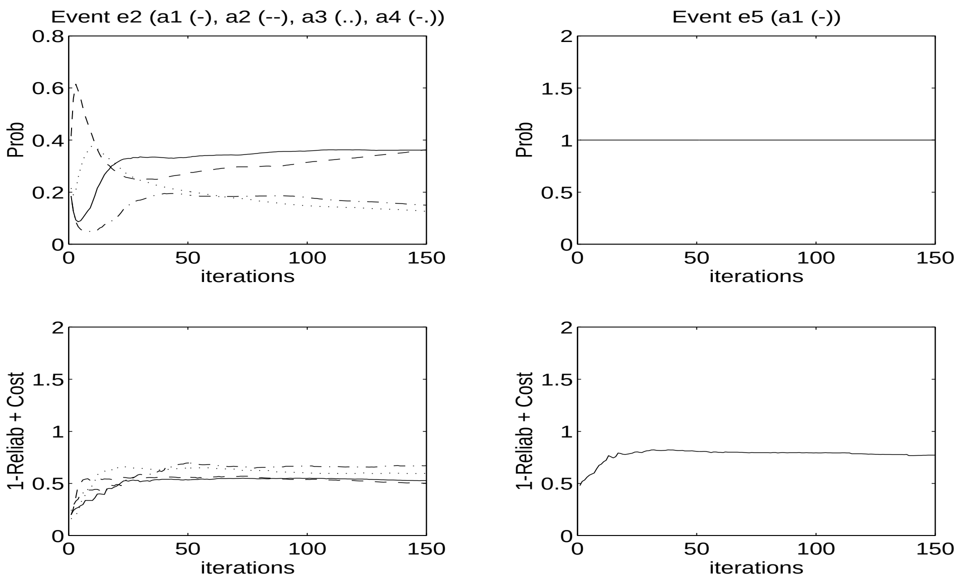

Figure 7.5: Evolution of probabilities and cost functions for algorithms translating events e_2 and e_5.

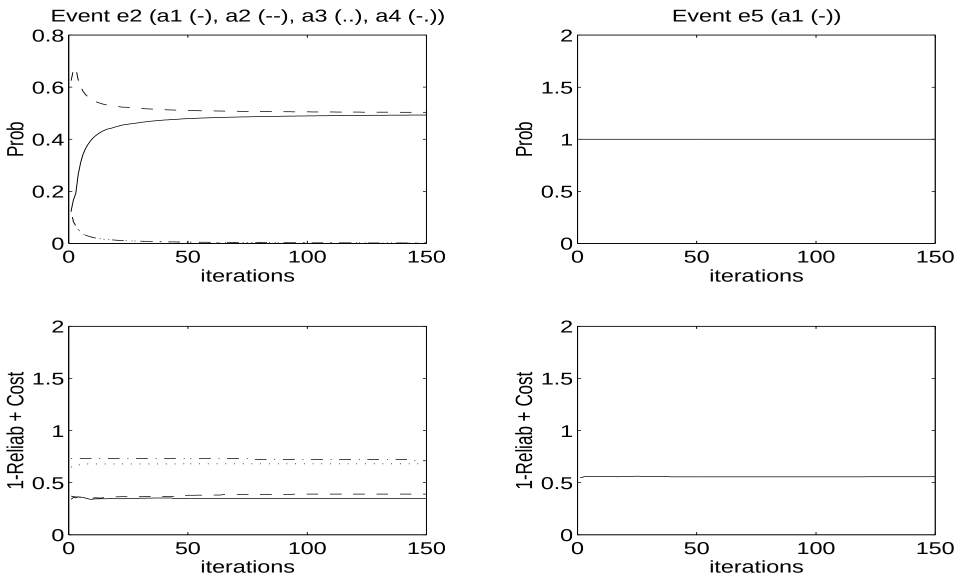

Figure 7.6: Evolution of probabilities and cost functions for algorithms translating events e_2 and e_5, when acceleration method is used.

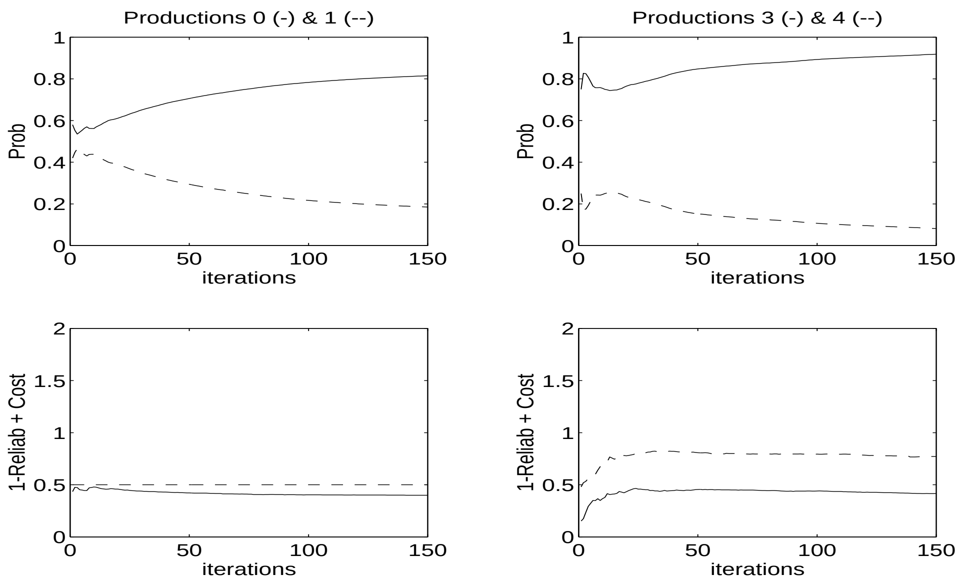

Figure 7.7: Evolution of probabilities and cost functions for productions 0,1,3 and 4.

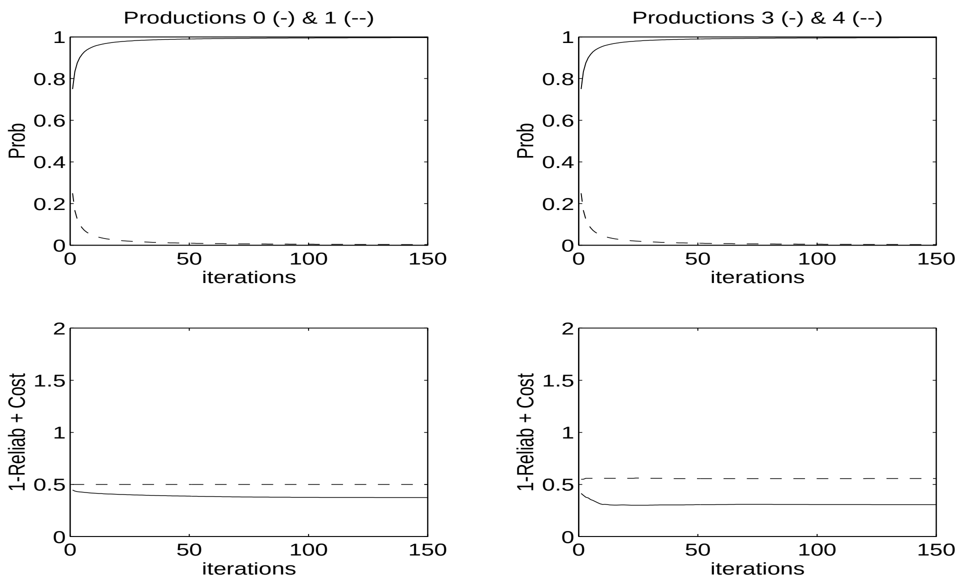

Figure 7.8: Evolution of probabilities and cost functions for productions 0,1,3 and 4, when acceleration method is used.

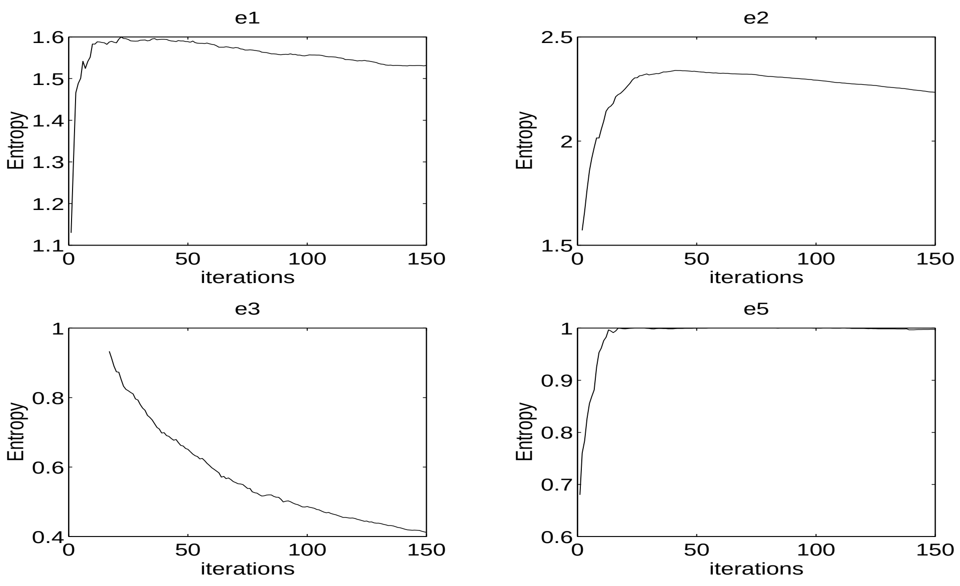

Figure 7.9: Evolution of entropies of events e_1, e_2, e_3 and e_5.

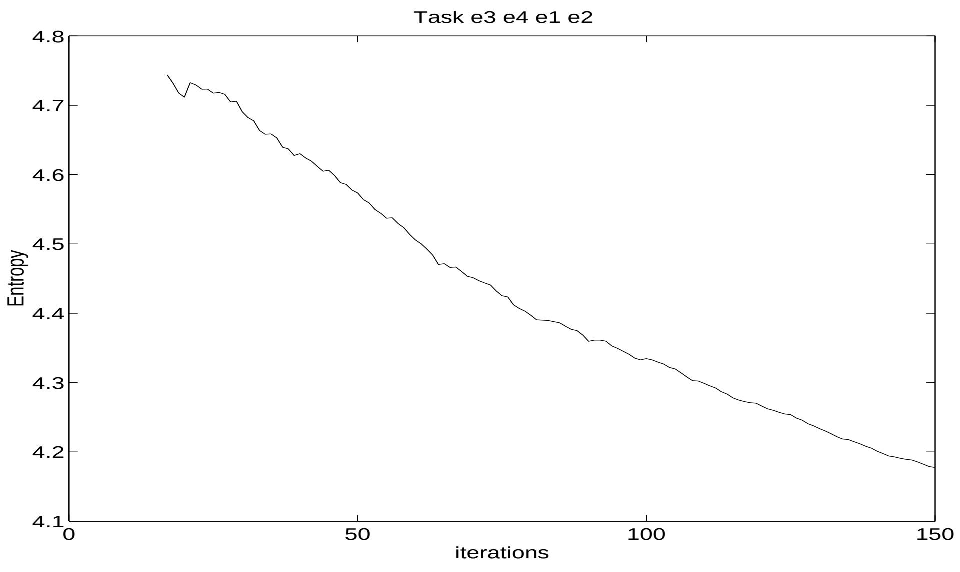

Figure 7.10: Evolution of entropy for task e_3 e_4 e_1 e_2.

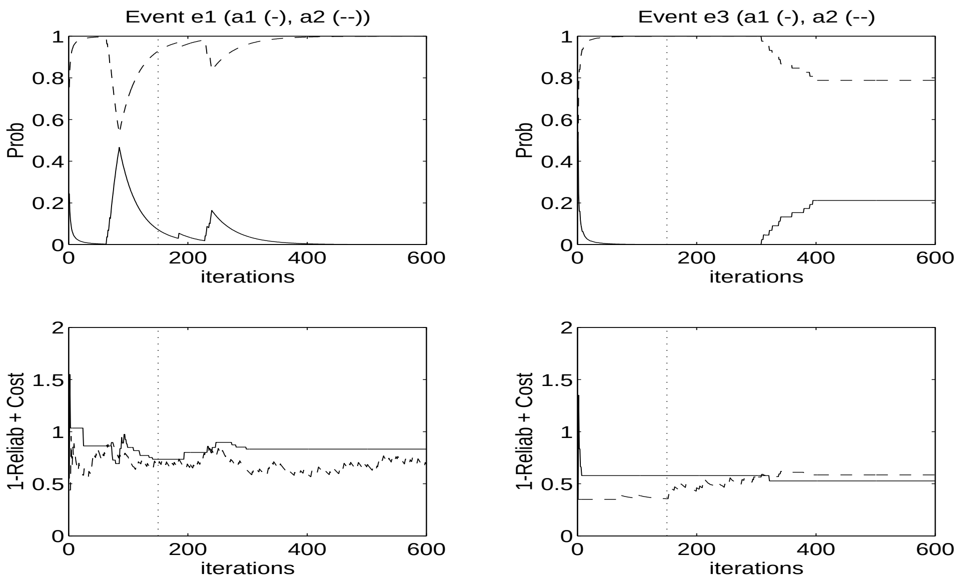

Figure 7.11: Evolution of probabilities and cost functions for algorithms translating events e_1 and e_3, with a change in the state of the environment.

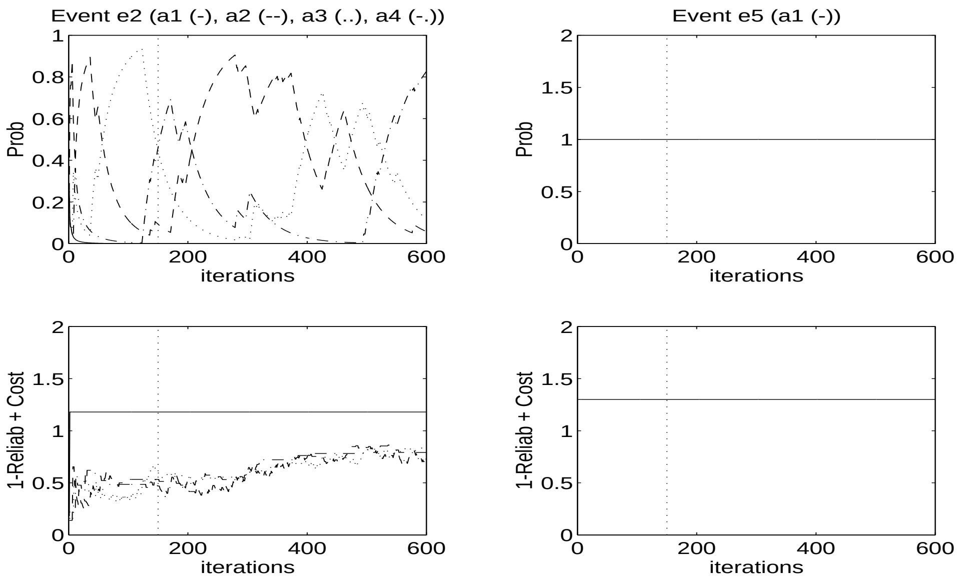

Figure 7.12: Evolution of probabilities and cost functions for algorithms translating events e_2 and e_5, with a change in the state of the environment.

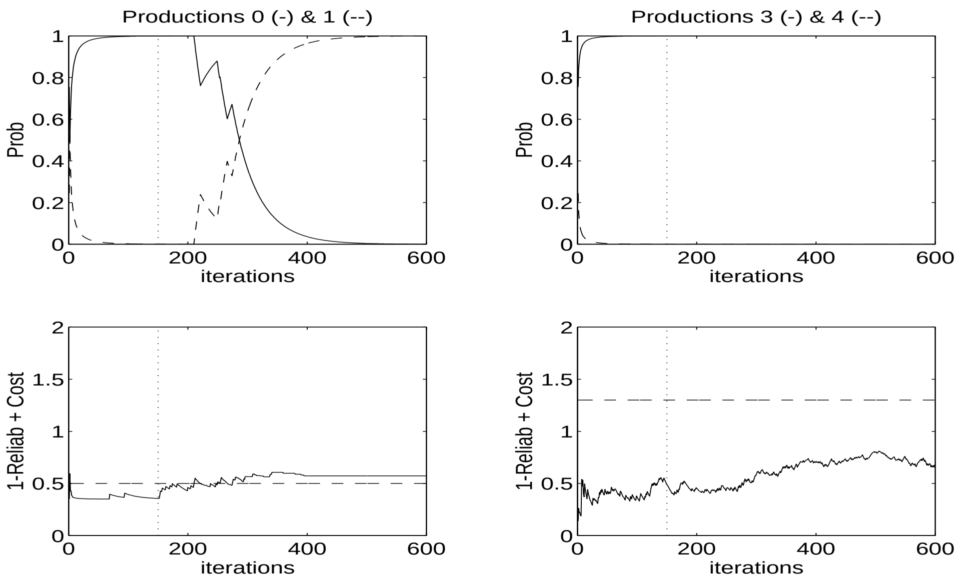

Figure 7.13: Evolution of probabilities and cost functions for productions 0,1,3 and 4, with a change in the state of the environment.

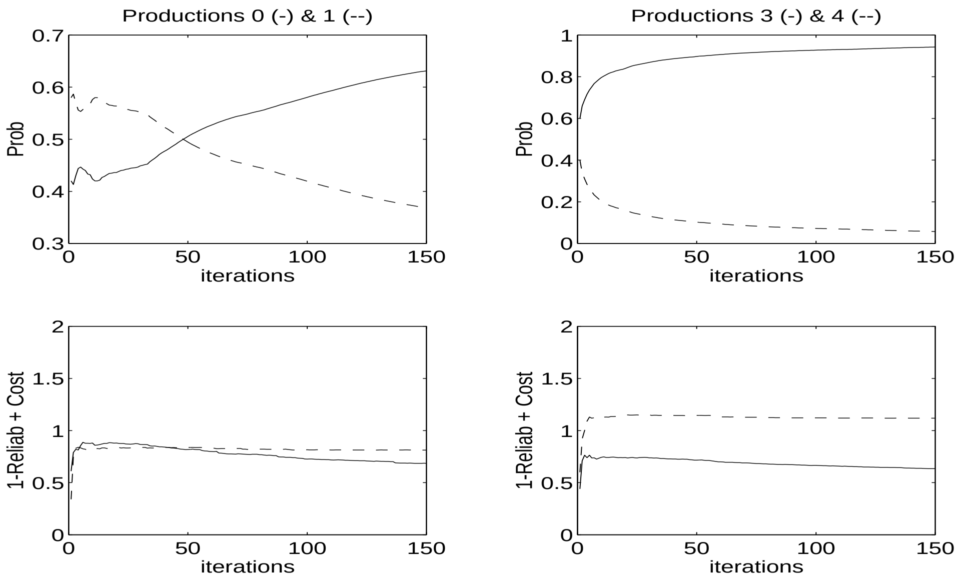

Figure 7.14: Evolution of probabilities and cost functions for productions 0,1,3 and 4, with direct task reinforcement.

CHAPTER 8
Conclusions and Future Work

The translation interfaces of a *Hierarchical Goal-Oriented Intelligent Machine*, based on the 3-level architecture proposed by Saridis[67], have been modeled as a 2-stage *Hierarchical Learning Stochastic Automaton* (HLSA). The model takes into account the feedback from the environment where the HGDIM operates, using that feedback to *learn* the *optimal task* among those capable of translating a *command* sent to the machine and the *optimal primitive algorithms* among those capable of translating the *primitive events* composing a task. An optimal action has been defined as the action (task or primitive algorithm) which *minimizes* a *cost function* recursively updated through feedback. The cost function of an action has two terms: one is the *cost* of applying the action, and the other is the complement of the *reliability* of the action. Reliability of an algorithm has been defined as the *probability that the algorithm meets a set of specifications*, while cost of an algorithm is a general measure of the resources needed by the algorithm to accomplish that reliability. Resources may include, but are not limited to, CPU time, memory, number of processors or number of FLOPs.

This approach unifies the solution of the *Decision Making* and *Learning* problems in Intelligent Machines at all levels of a HGDIM, as the optimization of a global cost function. The cost function is general enough to be applied to different algorithm types, a common requirement to Intelligent Machines. Furthermore, it allows the comparison between differently designed HGDIMs, given a (set of) goal(s).

Learning is based on a hierarchical version of *Fu's Generalized Learning Stochastic Automaton*: at each step, the cost function estimate is updated at the bottom level by a *Stochastic Approximation Algorithm.* The current estimates of the cost function for all the algorithms translating a primitive event are used by another

Stochastic Approximation Algorithm to update the algorithm subjective probabilities. Then, those estimates are propagated bottom-up by a set of equations to obtain event and task reliabilities. The latter are used by still another Stochastic Approximation Algorithm to update the task subjective probabilities.

Decision Making is accomplished similarly at the two stages of the HLSA:

- Commands are translated into tasks by random selection among the current task subjective probabilities;
- Primitive events are translated into primitive algorithms by random selection among the current subjective probabilities of primitive algorithms.

It was shown in Chapter 5 that the translation from commands to tasks is implemented by a *stochastic grammar*. Hence, the probabilities of selecting alternative productions of the grammar, not the probabilities of selecting tasks, are actually updated.

The *reinforcement scheme* proposed for the HLSA has been proven to select the optimal actions at the two levels *with probability 1* when the number of iterations tends to infinity.

8.1 Contributions

The contributions of this work are the following:

- A coherent analytical measure of algorithm cost and reliability. This measure can be improved on-line through feedback or used for off-line design of HGDIMs.
- A new measure of performance for HGDIMs, represented by a global cost function which combines cost and reliability;

- An original hierarchical version of Fu's generalized LSA: the Hierarchical Reinforcement Learning Scheme which uses feedback from the environment to recursively improve the estimate of the HGDIM cost function;
- The original modeling of a HGDIM by a HLSA that uses the Hierarchical Reinforcement Learning Scheme;
- The introduction of a Design Methodology for a HGDIM, based on the HLSA model;
- An original analytical study of the convergence rate of Fu's generalized LSA;
- A study of algorithms for acceleration of stochastic approximation in the context of this work;
- The simulation of two case studies describing the application of the formalism to two different areas of Intelligent Control: *intelligent process control* and *intelligent robotic systems*. The latter includes a realistic simulation of a PUMA manipulator, a stereo vision system and compliance control.

8.2 Future Work

The following are topics of future work aiming to extend the results described in this thesis:

- The implementation of the case studies using real setups;
- Further investigation of the behavior of a HGDIM in the presence of non-stationary environments. This should include the study of learning algorithms capable of adapting to changes in state of the environment without needing one LSA per state;

- Further investigation of methods to accelerate convergence speed of Stochastic Approximation algorithms. There are other methods described in the literature (e.g. Kesten's algorithm[27], or 2nd order stochastic approximation algorithms[66]) and other combinations of the methods presented in this work are possible, such as extending the acceleration to the probability update algorithm. The 2nd order stochastic approximation algorithm consists of using the covariance matrix of the estimated vector of parameters instead of the $\frac{1}{n+1}$ gain used in this work. This is equivalent in the estimation of the reliabilities (scalar case), but the convergence rate of the probability update algorithm may be increased by one order of magnitude (thus the name 2nd order).

- Analytical study of convergence when task probabilities are directly reinforced by the learning scheme, as shown in chapter 7;

- Study and proposal of new architectures for State of the Environment Feedback (see section 5.5). These may include a quantification of the uncertainty involved in modeling a stochastic environment by a finite number of states (for example a Markov chain);

- Study of reliability "indirect learning" methods which use existent reliability models and update some of their parameter estimates, such as covariance matrices, instead of directly updating the reliability estimate.

- The generalization of stochastic regular grammars used here to stochastic *context-free* or even *context-sensitive* grammars. This will permit dealing with more complex systems, requiring the derivation of tasks with higher level of sophistication.

8.3 Conclusion

This thesis represents a step towards a comprehensive analytical theory of Intelligent Controllers and Intelligent Systems, offering a framework for design and performance analysis of HGDIMs and showing its successful application to realistic simulations of systems as diverse as a Glass Melting Furnace and an Intelligent Robotic System.

LITERATURE CITED

[1] Y. S. Abu-Mostafa. Complexity of random problems. In *Complexity in Information Theory*. Springer-Verlag, 1986.

[2] AIMBURN. Final report of E.E.C. ESPRIT project 2192 - Advanced Intelligent Multi-Sensor System for Control of Boilers and Furnaces, 1992. M. Heitor, editor.

[3] J. S. Albus. Outline for a Theory of Intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), may/june 1991.

[4] B. Armstrong, O. Khatib, and J. Burdick. The explicit dynamic model and inertial parameters of the puma 560 arm. In *Proceedings of 1986 IEEE International Conference on Robotics and Automation*, pages 510–518, 1986.

[5] N. Baba. *New Topics in Learning Automata Theory and Applications*. Lecture Notes in Control and Information Science, Springer-Verlag, 1985.

[6] D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.

[7] A. G. Barto and P. Anandan. Pattern-recognizing stochastic learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(3):360–375, 1985.

[8] R. Beard and G. N. Saridis. A cost measure for efficient scheduling in Intelligent Machines. In *Proceedings of 8th International Symposium on Intelligent Control*, Aug 1993.

[9] W. L. Brogan. *Modern Control Theory*. Prentice Hall, 1991. 3rd Edition.

[10] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 1986.

[11] M. G. Carvalho and M. Nogueira. Glass quality evaluation via 3-d mathematical modeling of glass melting furnaces. In *Proc. E. S. G. Conference on the Fundamentals of the Glass Manufacturing Process*, Sep 1991. Shefield, UK.

[12] M. G. Carvalho, V. S. Semiao, and P. J. Coelho. Modelling and optimization of the NO formation in an industrial glass furnace. *Journal of Engineering for Industry - Trans. ASME*, 114:514–523, Nov 1992.

[13] J. J. Craig. *Introduction to Robotics*. Addison-Wesley, 1985.

[14] A. Desrochers, editor. *Intelligent Robotic Systems for Space Exploration.* Kluwer Publishers, 1992.

[15] A. Dvoretzky. On Stochastic Approximation. In *Proc. of 3rd Berkeley Symposium of Math. Stat. and Prob.*, volume 1. University of California Press, Los Angeles, 1956.

[16] E. D. Farmer and G. F. Bryant. The design of integrated control schemes for the glass-making industry. In *Proc. 7th Annual ESPRIT CIM/Europe Conference*, May 1991. Turin.

[17] K. S. Fu. Learning Control, Systems - review and outlook. *IEEE Transactions on Automatic Control*, AC-15(2), 1970.

[18] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey – part II. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-5(4), 1975.

[19] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *Robotics: Control, Sensing, Vision and Intelligence.* McGraw Hill, 1987.

[20] K. S. Fu and J. M. Mendel. *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications.* Academic Press, 1970.

[21] K. S. Fu and Z. J. Nikoliç. On some reinforcement techniques and their relation to the stochastic approximation. *IEEE Transactions on Automatic Control*, AC-11(2):756–758, 1966.

[22] K. S. Fu, Z. J. Nikoliç, Y. T. Chien, and W. G. Wee. On the stochastic approximation and related learning techniques. Technical Report TR-EE66-6, Purdue University - School of Electrical Engineering, Lafayette, Indiana, 1966.

[23] J. H. Graham and G. N. Saridis. Linguistic decision structures for hierarchical systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-12:323–333, 1982.

[24] N. Hogan. Impedance control: an approach to manipulation: parts 1,2,3. *Journal of Dynamic Systems, Measurement and Control*, 107:1–24, March 1985.

[25] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop. Neural networks for control systems - a survey. *Automatica*, 28(6):2083–1112, 1992.

[26] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106(4), 1957.

[27] H. Kesten. Accelerated stochastic approximation methods. *Ann. Math. Stat.*, 29(1), 1958.

[28] P. J. King and E. H. Mandani. The application of fuzzy control systems to industrial processes. *Automatica*, 13:235–242, 1977.

[29] R. E. King and F. C. Karonis. Multi-level expert control of a large-scale industrial process. *Fuzzy Computing*, pages 323–339, 1988.

[30] A. N. Kolmogorov. Combinatorial foundations of Information Theory and the calculus of probability. *Russian Mathematical Surveys*, 38(4):29–40, 1983.

[31] B. C. Kuo. *Automatic Control Systems.* Prentice Hall, 1987.

[32] K. Kyriakopoulos and G. N. Saridis. Minimum jerk path generation. In *Proceedings of 1988 IEEE International Conference on Robotics and Automation*, pages 364–369, 1988.

[33] S. Lakshmivarahan and M. A. L. Thathachar. Bayesian learning and reinforcement schemes for stochastic automata. In *Proceedings of International Conference on Cybernetics and Society*, 1972.

[34] S. Lakshmivarahan and M. A. L. Thathachar. Absolutely expedient learning algorithms for stochastic automata. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-3, May 1973.

[35] P. M. Larsen. *Industrial Applications of Fuzzy Logic.* 1988.

[36] S. Lee and Y. Kay. An Accurate Estimation of 3-D Position and Orientation of a Moving Object for Robot Stereo Vision: Kalman Filter Approach. In *Proceedings of 1990 IEEE International Conference on Robotics and Automation*, pages 414–419, 1990.

[37] D. R. Lefebvre and G. N. Saridis. A computer architecture for Intelligent Machines. In *Proceedings of 1992 IEEE International Conference on Robotics and Automation*, may 1992.

[38] F. L. Lewis. *Optimal Estimation.* John Wiley and Sons, 1986.

[39] M. Li and P. Vitányi. Kolmogorov Complexity and its applications. In *Handbook of Theoretical Computer Science*, pages 187–254. North-Holland Amsterdam, 1990.

[40] P. Lima, P. Oliveira, C. Ferreira, and J. Sentieiro. The architecture of a knowledge based controller for the operation of glass melting furnaces. In *Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control*, may 1991.

[41] P. U. Lima and R. Beard. Using neural networks and Dyna algorithm for integrated planning, reacting and learning in systems. Technical Report CIRSSE-122, Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1992.

[42] P. U. Lima and G. N. Saridis. Measuring Complexity of Intelligent Machines. In *Proceedings of 1993 IEEE Int. Conf. Robotics and Automat.*, may 1993.

[43] P. U. Lima and G. N. Saridis. Hierarchical reinforcement learning and decision making for Intelligent Machines. In *Proceedings of 1994 IEEE Int. Conf. Robotics and Automation*, May 1994.

[44] P. U. Lima and G. N. Saridis. Intelligent process control based on hierarchical reinforcement learning. In *Proc. of 2nd. IEEE Mediterranean Symp. on New Directions in Control & Automation*, June 1994.

[45] P. U. Lima and G. N. Saridis. Performance improvement of Autonomous Underwater Vehicles based on hierachical reinforcement learning. In *Proceedings of 1st ISR Workshop on Autonomous Underwater Vehicles.* Kluwer, 1994.

[46] P. U. Lima and G. N. Saridis. A performance measure for Intelligent Machines based on complexity and reliability. In *Proceedings of SY.RO.CO 94*, September 1994.

[47] Long-Ji Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *Proceedings of International Conference Simulation of Adaptive Behavior.* Cambridge,MA:The MIT Press, 1991.

[48] G. L. Luo and G. N. Saridis. Optimal/PID formulation for control of robotic manipulators. *IEEE Journal of Robotics and Automation*, 1(3), December 1985.

[49] J. McInroy and G. Saridis. Techniques for selecting pose algorithms. *to be published in Automatica*, 1994.

[50] J. E. McInroy and G. N. Saridis. Reliability analysis in Intelligent Machines. *IEEE Transactions on Systems, Man and Cybernetics*, 20(4), 1990.

[51] J. E. McInroy and G. N. Saridis. Entropy searches for robotic reliability assessment. In *Proceedings of 1993 IEEE Int. Conf. Robotic Automat.*, May 1993.

[52] R. W. McLaren. A Stochastic Automaton model for the synthesis of Learning Systems. *IEEE Transactions on Systems Science and Cybernetics*, SSC-2:109–114, 1966.

[53] A. Meystel. *Autonomous Mobile Robots.* World Scientific, 1991.

[54] M. C. Moed and G. N. Saridis. A Boltzmann machine for the organization of intelligent machines. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-4(4):323–334, sep 1990.

[55] J. Musto and G. N. Saridis. An entropy-based reliability assessment technique for intelligent machines. In *Proceedings of 8th International Symposium on Intelligent Control*, Aug 1993.

[56] J. C. Musto. A entropy-based reliability analysis for Intelligent Machines. Technical Report CIRSSE-132, Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1992.

[57] K. S. Narendra and K. Parthasarathy. Identification and control for dynamic systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[58] K. S. Narendra and M. A. L. Thathachar. Learning Automata - a survey. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-4(4), 1974.

[59] K. S. Narendra and M. A. L. Thathachar. *Learning Automata - an Introduction.* Prentice Hall, 1989.

[60] Z. J. Nikoliç and K. S. Fu. An algorithm for learning without external supervision and its application to learning control systems. *IEEE Transactions on Automatic Control*, AC-11(13), 1966.

[61] A. Papoulis. *Probability, Random Variables and Stochastic Processes.* NY:McGraw-Hill, 1965.

[62] R. P. Paul. *Robot Manipulators: Mathematics, Programming and Control.* MIT Press, 1981.

[63] A. G. Pincus. *Melting Furnace Operation in the Glass Industry.* Books for Industry and Glass Industry Magazine, 1980.

[64] S. Ross. *A First Course in Probability.* Macmillan Publ. Co., 1988. 3rd. Edition.

[65] Michael Ryan. Implementation of robotic force control with position accommodation. Master's thesis, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1992.

[66] G. N. Saridis. *Self-Organizing Control of Stochastic Systems.* Marcel Dekker, Inc, 1977.

[67] G. N. Saridis. Toward the realization of intelligent controls. *IEEE Proceedings*, 67(8), 1979.

[68] G. N. Saridis. Entropy formulation for optimal and adaptive control. *IEEE Transactions on Automatic Control*, 33(8):713–721, 1988.

[69] G. N. Saridis. Analytic formulation of the IPDI for Intelligent Machines. *Automatica*, 25(3):461–467, 1989.

[70] G. N. Saridis. Architectures for Intelligent Machines. Technical Report CIRSSE-58, Center for Intelligent Robotic Systems for Space Exploration (CIRSSE), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1991.

[71] G. N. Saridis and J. H. Graham. Linguistic decision schemata for intelligent robots. *Automatica*, pages 121–126, 1984.

[72] G. N. Saridis and Harry E. Stephanou. A hierarchical approach to the control of a prosthetic arm. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-7(6):407–420, 1977.

[73] G. N. Saridis and K. P. Valavanis. Analytical design of Intelligent Machines. *Automatica*, 24:123–133, 1988.

[74] R. S. Sutton. Learning to predict by the methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988.

[75] R. S. Sutton. First results with Dyna, an integrated architecture for learning, planning and reacting. In *Neural Networks for Control*. The MIT Press, 1990.

[76] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning in direct adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22, 1992.

[77] Technical Committee on Intelligent Control. Report of task force on Intelligent Control, IEEE Control Systems Society. *IEEE Control Systems Magazine*, 14(3), June 1994. P. Antsaklis, editor.

[78] M. A. L. Thathachar and K. R. Ramakrishnan. A hierarchical system of Learning Automata. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(3), 1981.

[79] M. A. L. Thathachar and P. S. Sastry. A new approach to the design of reinforcement schemes for Learning Automata. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(1):168–175, 1985.

[80] J. Traub. Introduction to Information-Based Complexity. In *Complexity in Information Theory*, pages 62–76. Springer-Verlag, 1986.

[81] J. Traub, G. Wasilkowsky, and H. Woźniakowsky. *Information-Based Complexity*. Academic Press, Inc., 1988.

[82] J. Traub and H. Woźniakowsky. *A General Theory of Optimal Algorithms*. Academic Press, Inc., 1980.

[83] M. L. Tsetlin. On the behavior of Finite Automata in random media. *Avtomatika i Telemekhanika*, 22(10), 1961.

[84] J. N. Tsitsiklis. Complexity-theoretic aspects of problems in control theory. Keynote speech in 31st IEEE Conference on Decision and Control, Dec 1992.

[85] Ya. Z. Tsypkin. *Foundations of the Theory of Learning Systems.* Academic Press, 1973. Translation by Z. J. Nikoliç.

[86] Ya. Z. Tsypkin and A. S. Poznyak. Learning Automata. *Journal of Cybernetics and Information Science*, 1(1), 1977.

[87] Unimation Robotics. *Unimate PUMA Robot*, April 1980. Text 398H1A, Vol. 1.

[88] K. P. Valavanis and G. N. Saridis. *Intelligent Robotic Systems.* Kluwier Publishers, 1992.

[89] V. I. Varshavskii and I. P. Vorontsova. On the behavior of Finite Automata with stochastic structure. *Avtomatika i Telemekhanika*, 24(3), 1963.

[90] J. A. Victor, J. P. Costeira, J. A. Tomé, and J. J. Sentieiro. A computer vision system for the characterization and classification of flames in glass furnaces. *IEEE Trans. on Industry Applications*, 29(3):470–479, May-June 1993.

[91] M. D. Waltz and K. S. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, AC-10(4):390–398, 1965.

[92] F. Y. Wang. *A Coordinatory Theory for Intelligent Machines.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1990.

[93] Fei-Yue Wang and G. N. Saridis. A coordination theory for Intelligent Machines. *Automatica*, 26(5):833–844, 1990.

[94] Lofti A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(1), jan 1973.

[95] G. Zames. On the metric complexity of causal linear systems, ϵ-entropy and ϵ-dimension for continuous time. *IEEE Transactions on Automatic Control*, AC-24(4):220–230, 1979.

APPENDIX A
Simulation of Case Study 2 - Intelligent Robotic System

This appendix describes the implementation details of case study 2 (Intelligent Robotic System) simulation. The description covers the setup of the workspace, algorithm implementation and cost determination for the main events, world model structure and software issues regarding the communication between the Intelligent Machine and the environment, separately simulated. The simulation was designed to balance accuracy and realism with reasonable computational load and development time. The details given here will help guiding a future implementation of the case study.

A.1 Setup of the Workspace

A general picture of the simulated workspace is depicted in Figure 7.2. Also shown are the *camera coordinate frame, world coordinate frame* and *table coordinate frame*. The origin of the *strut coordinate frame*, not shown, is located in the middle top of the strut. X_s lies along the length of the strut and Z_s points downwards. The PUMA coordinate frames are those used in [4]. The world coordinate frame coincides with coordinate frame 1 when the PUMA is in the *zero position* (see Figure 7.2). The origin of the camera coordinate frame is located halfway between the origins of the two image plane coordinate frames, at a distance λ along Z (see Figure A.2).

The strut is on a table located inside the workspace of the PUMA 560 (see Figures A.1 and A.2. Actual dimensions were taken from [87]). Strut and PUMA gripper dimensions are shown in Figure A.3.

The two cameras are positioned above the workspace, and their image coordinate frames are perfectly aligned, differing only in the origin of their coordinates by a distance called *baseline* (B). The image planes are also parallel to the table top.

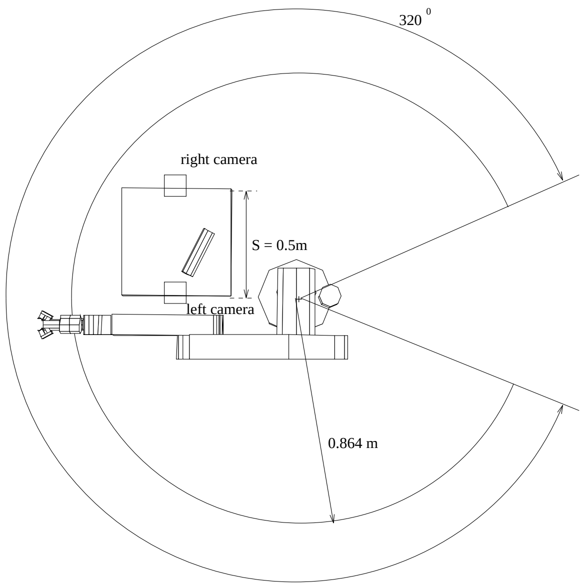

Figure A.1: Above view of the workspace showing PUMA 560 operating envelope.

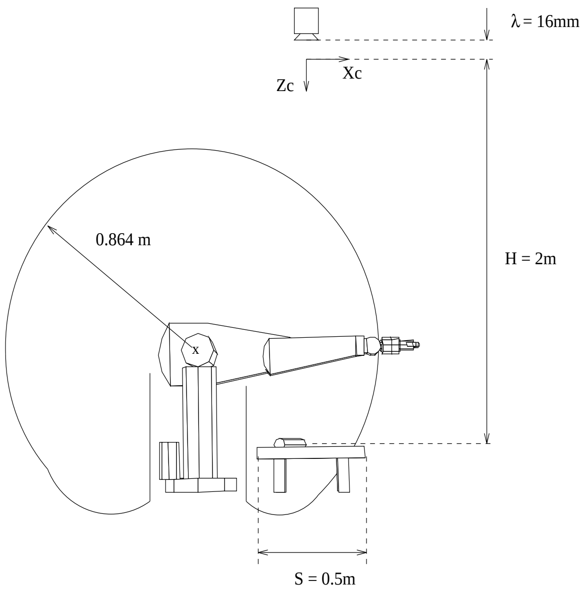

Figure A.2: Lateral view of the workspace showing PUMA 560 operating envelope.

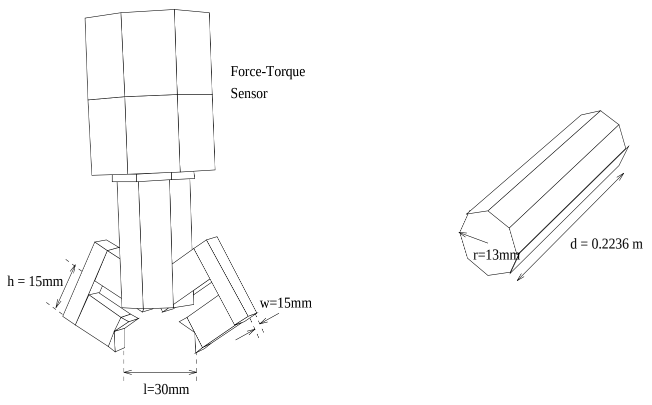

Figure A.3: Gripper and strut dimensions.

The camera produces images of 512 by 512 pixels. The pixel width pw and focal length λ are given:

$$pw = 10\mu m \tag{A.1}$$

$$\lambda = 16mm \tag{A.2}$$

The vertical distance H between the top of the table and the image plane was dimensioned based on the the camera parameters and such that the whole top of the square table can seen by the cameras (see figures for meaning of symbols):

$$H \geq \frac{\lambda S}{512pw} = 1.5625m$$

Hence, H was made equal to 2 meters.

The baseline distance is determined from the desired precision for depth estimates ΔZ in the camera coordinate frame by the equation

$$|\Delta Z| = \frac{H^2}{\lambda B}|\Delta\delta|$$

where δ is the *disparity along* X, or the difference between the pixel coordinates in the two cameras corresponding to the same 3D point in the camera frame, and $\Delta\delta$ its (finite) precision. Assuming $\Delta\delta = 1\text{pixel} = 10\mu m$ and $\Delta Z = 5mm$, $B = 0.5m$.

Some important homogeneous transformation matrices representing relative poses of coordinate frames follow. The subscripts and superscripts used are s for strut, t for table, w for world and c for camera. T_c^w denotes the pose of the camera coordinate frame with respect to the world frame.

$$T_t^w = \begin{pmatrix} 1 & 0 & 0 & 0.3 \\ 0 & 1 & 0 & -0.3 \\ 0 & 0 & 1 & -0.46 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.3}$$

$$T_s^t = \begin{pmatrix} 0.45 & 0.89 & 0 & 0.15 \\ 0.89 & -0.45 & 0 & 0.3 \\ 0 & 0 & -1 & 0.03 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.4}$$

$$T_s^w = \begin{pmatrix} 0.45 & 0.89 & 0 & 0.45 \\ 0.89 & -0.45 & 0 & 0 \\ 0 & 0 & -1 & -0.43 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.5}$$

$$T_c^w = \begin{pmatrix} 1 & 0 & 0 & 0.55 \\ 0 & -1 & 0 & -0.05 \\ 0 & 0 & -1 & 1.54 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.6}$$

(A.7)

A.2 Software Issues and Structure of World Model

The HLSA model of the IM was written in C, including all necessary data structures and user interface. Each of the events was written as a separate Matlab `m-file`. The world model is initialized by another `m-file`, and includes object dimensions, modified Denavit-Hartenberg description of the PUMA manipulator[13], initial *zero position* of manipulator, transformation between coordinate frames, initial rough estimate of strut pose (obtained by adding noise to the joint positions corresponding to the actual pose), nominal image coordinates of the strut extremes in the two cameras, orientation and position clearance of the manipulator tool. The communication between C and Matlab used the Computational Engine Services provided by version 4.1 of Matlab.

Each m-file representing an event has approximately the same structure:

function header - function [pr]=*eventname*(algnumber), where pr represents the returned success or failure signal, and algnumber is the number designating the algorithm selected to translate the event.

global variables - declared as global *varname[s]*. All global variables are shared among m-files and are declared as global inside the module which initializes the world model. They include the current state of the environment, desired pose of manipulator tip and current pose of manipulator tip. The set of global variables represent the world model. All events can update and access the world model using them.

function body - generally divided in if ... then modules corresponding to the different combinations of current state of the environment and selected algorithm. In some cases, the behavior of the algorithm does not depend on the current state of the environment, hence no conditional tests to check the state of the environment are necessary. Each submodule must include the assignment of a success or failure value to the variable pr.

A.3 Events Simulation

e_3: locate object

Objective: Assign to the global variable Ttipfinal the estimated pose of the strut coordinate frame. If event e_3 is not invoked (production 1 is selected), the default value assigned to Ttipfinal is the rough estimate of the strut pose.

Specifications: If $|\hat{Z}_1^c - \hat{Z}_2^c| < 5mm$, report success; otherwise, report failure. $\hat{Z}_i^c$, $i = 1, 2$ is the estimated Z coordinate of the strut in the camera coordinate frame (see below). Recall that the distance from the strut to the camera frame is not known, but it is assumed that the strut lies in plane parallel to the image planes of the two cameras.

Algorithms: Two different algorithms may be selected: the open loop (a_1^3) and closed loop (a_2^3) algorithms in chapter 4. Both estimate the coordinates of the strut extreme points $(\underline{P}_i^{l,r} = (X_i^{l,r}\ Y_i^{l,r}\ Z_i^{l,r})^T\ i = 1, 2)$ in the left camera l and right camera r. The *pin-hole parallel stereo model*[36] is used to obtain the transformation between image plane coordinates in the two cameras and 3D coordinates in the camera coordinate frame:

$$X_i^c = \frac{BX_i^{l,r}}{\delta_i} \tag{A.8}$$

$$Y_i^c = \frac{B(Y_i^l + Y_i^r)}{2\delta_i} \tag{A.9}$$

$$Z_i^c = \frac{B\lambda}{\delta_i},\ i = 1, 2;\ \delta_i = Y_i^l - Y_i^r \tag{A.10}$$

The open loop and closed loop algorithms are not actually implemented, but their performance in the presence of spot noise and finite pixel resolution is simulated, according to the results in section 4.4.3.5 and [51].

The general algorithm for this event, distinguished by selected algorithm and state of the environment at step 2, is:

1. Add noise with uniform distribution in the interval $[-pw/2, pw/2]$ to the nominal image coordinates of the strut extremes in the two cameras, for open loop algorithm only. This simulates errors due to pixel resolution, unexistent in the closed loop algorithm.

2. To simulate estimate errors due to spot noise, add or subtract (with probability 0.5) 1 pixel to the image coordinates obtained in step 1. On the average, do this:

   - for a_1^3, 20% of the runs when lights are on and 40% of the runs when lights are off;
   - for a_2^3, 5% of the runs when lights are on and 30% of the runs when lights are off;

3. Determine the position estimate of the strut extremes $\hat{P}_i^c = (\hat{X}_i^c\ \hat{Y}_i^c\ \hat{Z}_i^c)^T$, $i = 1, 2$ from Equations (A.8-A.10), and $\hat{P}_i^t = T_c^t \hat{P}_i^c$.

4. Check specifications. Make $\hat{Z}_1^t = \hat{Z}_2^t = (\hat{Z}_1^t + \hat{Z}_2^t)/2$.

5. Determine

$$\begin{aligned} \hat{d} &= \sqrt{(\hat{X}_1^t - \hat{X}_2^t)^2 + (\hat{Y}_1^t - \hat{Y}_2^t)^2} \\ \hat{\theta} &= \arctan(\frac{\hat{Y}_2^t - \hat{Y}_1^t}{\hat{X}_2^t - \hat{X}_1^t}) \end{aligned}$$

6.

$$\hat{T}_s^t = \text{TRANS}(\underline{\hat{P}}_1^t)\ \text{ROT}(X_t, \pi)\ \text{ROT}(Z_s, -\hat{\theta})\ \text{TRANS}((\hat{d}/2\ 0\ 0))$$

   where TRANS and ROT are the homogeneous translation and rotation matrices described in [19].

7. Ttipfinal $\leftarrow \hat{T}_s^w = T_t^w \hat{T}_s^t$

Cost: If the cost is made proportional to the number of frames averaged to obtain the image processed by the algorithms, and if both algorithms use the same number of averaged frames, both will have the same cost (see Table 7.2).

e_1: move manipulator

Objective: Move the manipulator from its *zero position* to a point in space $10cm$ above the strut location estimated by e_3. Hence, the final desired pose of the manipulator gripper is given by

$$\text{TRANS}((0\ 0\ 0.1))\ \text{Ttipfinal}$$

.

Specifications: Given the desired final position $(x_d(t_f)\ y_d(t_f)\ z_d(t_f))$ and the actual final position of the manipulator $(x(t_f)\ y(t_f)\ z(t_f))$, if

$$(x(t_f) - x_d(t_f))^2 + (y(t_f) - y_d(t_f))^2 + (z(t_f) - z_d(t_f))^2 < 10\mu m$$

report success; otherwise report failure.

Algorithms: There are two algorithms to translate this event. Both are controllers based on the Computed Torque Method (see section 4.4.1). The choice of gains k_p and k_v distinguishes them:

Algorithm	k_p	k_v	w_n	ξ
a_1^1	100	10	10	0.5
a_2^1	100	8	10	0.4

Algorithm a_2^1 is more underdamped and after some overshoot reaches the final value faster. Since only the final position is checked and not the deviations from the nominal trajectory, a_2^1 is expected to be more reliable than a_1^1.

The PUMA dynamics was simulated based on the equations and estimated Modified Denavit-Hartenberg parameters for the PUMA 560 in [4]. The inverse kinematics came from notes written by Ken Kreutz-Delgado while at JPL in Pasadena, California. The software used was written by S. Murphy and D. Swift at CIRSSE-RPI.

Uncertainty is due to three factors:

- The controller uses a time-invariant and diagonalized Mass matrix;
- Except for gravity, non-linear terms are not subtracted in the control law;
- Noise with variance 10^{-5} was added to the actual joint positions.

Reliability is also influenced by the velocity of the manipulator tip. In the simulation, this velocity was made equal to $0.6ms^{-1}$, which is less than the nominal maximum velocity ($1ms^{-1}$) [87].

Cost: Using Equation (4.35), the normalized cost ρC of the two algorithms was defined as $(i = 1, 2)$:

$$\begin{aligned}\rho C(a_i^1) &= \frac{C(a_i^1)}{C(a_1^1) + C(a_2^1)} \\ &= \frac{K_{p_i} K_{v_i} \sigma_p^2 + K_{v_i} \sigma_v^2}{((K_{p_1} K_{v_1} + K_{p_2} K_{v_2})\sigma_p^2 + (K_{v_1} + K_{v_2})\sigma_v^2)}\end{aligned}$$

Replacing $\sigma_p^2 = 10^{-5}$ and $\sigma_v^2 = 0$ and the gains from the above table, the costs in Table 7.2 are obtained.

e_2: grasp object with compliance and e_5: grasp object hard

Objective: Move manipulator from current pose to Ttipfinal and grasp object. e_2 uses force control and e_5 uses position control.

Specifications: e_5 must pass two clearance tests: the *position clearance test* and the *orientation clearance test*, in this order. e_2 must pass a further test: if $|f_{z_d} - \bar{f}_z(N_2)| < 0.1N$ report success; otherwise report failure. f_{z_d} is the desired force and $\bar{f}_z(N_2)$ is the sample mean of actual force sensed from step $N_2 - 10$ to N_2 of the simulation (see below). A failure is reported if any of the above tests fails.

Algorithms: In the simulation of e_2, the Position Accommodation Controller described in section 4.4.2 is used. The manipulator tip is required to behave as a *Mass, Spring and Damper* (MSD) system. The integrators in Figure 4.1 were

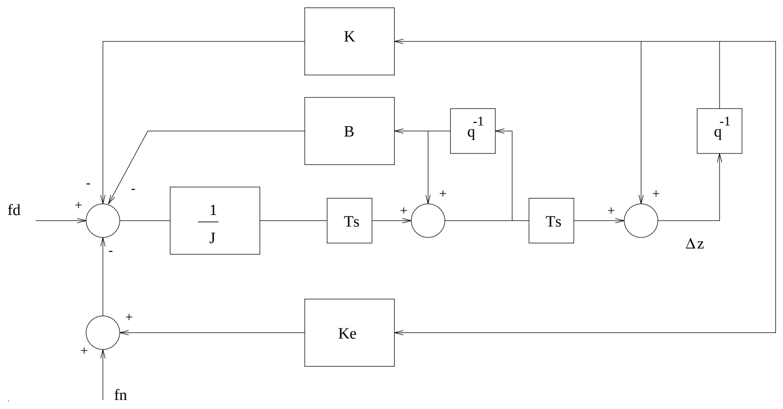

Figure A.4: Discrete Mass, Spring and Damper block diagram.

replaced by a first-order rectangular integration to obtain the implemented discrete version in Figure A.4.

In the figure, T_s is the sampling time of the trajectory generator. The timeout referenced in section 4.4.2 is the parameter N_2 above. In the simulation, $N_2 = 100$ and $T_s = 25ms$, which allows a maximum traverse time of $2.5s$ for a path length of $10cm$. This can be considered fine motion (velocity is much less than the maximum velocity of the manipulator tip), and errors of the manipulator position controller were disregarded, so the PUMA dynamics did not have to be simulated in this case. The figure also shows noise added to the force sensor. This noise was simulated as a zero mean gaussian random number generator with variance 0.01.

Algorithm	B	K	N_c	N_s
a_1^2	50	10	25	24
a_2^2	30	10	19	40
a_3^2	30	50	21	40
a_4^2	50	50	30	24

Table A.1: Parameterization and performance of e_2 primitive algorithms.

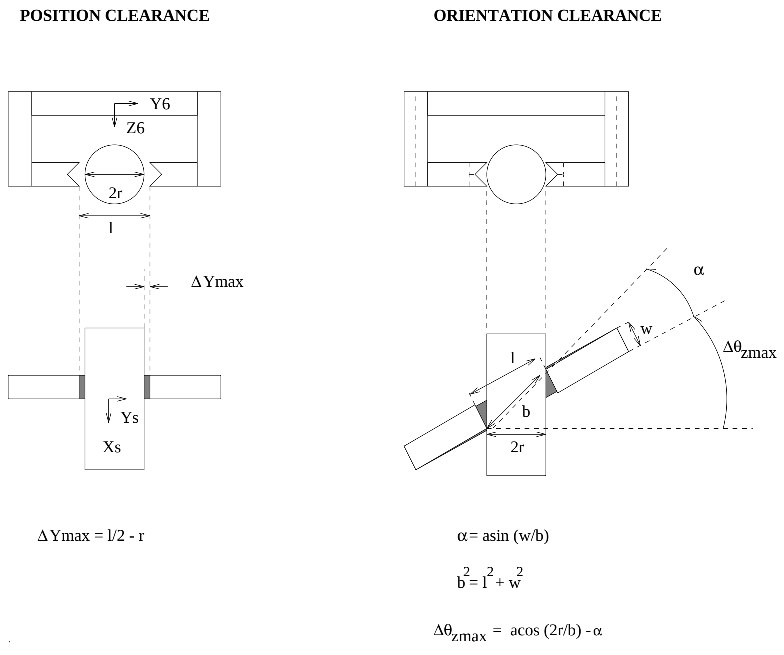

Figure A.5: Position and Orientation Clearances.

4 algorithms were considered to translate e_2, consisting of all combinations of two different values for the parameters K and B. The discrete MSD system was initially simulated with no force noise, to determine the number of steps N_c taken from the initial position to the point where the tip contacts with the strut, and the number of steps N_s taken from contact to the point where the sensed force comes within 5% of the desired force. All values are tabulated in Table A.1.

The position and orientation clearances are illustrated in Figure A.5. Given the strut and gripper dimensions in Figure A.3, the clearance values are:

$$\begin{aligned} \text{position: } \Delta Y_{\max} &= 2mm \\ \text{orientation: } \Delta\theta_{z\max} &= 12.6^o \end{aligned}$$

The clearance tests are made in two steps:

1. Given T_c^w and T_s^w, the homogeneous transformation matrices representing respectively the current pose of the manipulator tip and the strut pose, $\delta T =$

$(T_c^w)^{-1}T_s^w$ is determined. If the orientation errors are small, δT is given by:

$$\delta T = \begin{pmatrix} & \delta R & & \underline{\delta P} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $\underline{\delta P} = (\delta x\ \delta y\ \delta z)^T$ represents the position error vector and

$$\delta R = \begin{pmatrix} 1 & -\delta\theta_z & \delta\theta_y \\ \delta\theta_z & 1 & -\delta\theta_x \\ -\delta\theta_y & \delta\theta_x & 1 \end{pmatrix}$$

represents the rotation error[62], where $\delta\theta_x$, $\delta\theta_y$, $\delta\theta_z$ are the differential rotation errors about X, Y and Z, respectively.

2. To simulate compliance, extra position and orientation clearances are allowed to all four algorithms. The percentages of clearance increase depend on the parameter K and are tabulated below:

Algorithm	K	Δp_c	Δo_c
a_1^2, a_2^2	10	20 %	50 %
a_3^2, a_4^2	50	10 %	30 %

Depending on the algorithm selected, $\Delta Y_{\max}$ and $\Delta\theta_{z\max}$ are increased by the corresponding percentage, and compared with δy and $\delta\theta_z$ to check actual clearance.

Event e_5 is translated by only one algorithm. Only position is controlled. The number of steps until contact N_c was designed such that the movement is slower than when e_2 algorithms are applied: $N_c(a_1^5) = 40$. The extra position and clearance percentage are smaller than for e_2 algorithms, to reflect the existence of passive compliance only: $\Delta p_c = 0\%$, $\Delta o_c = 20\%$.

Cost: The number of steps until contact N_c of the four e_2 algorithms and the e_5 algorithm are put together to obtain normalized costs for the 5 algorithms:

$$\rho C(a_i^{2,5}) = \frac{N_c(a_i^{2,5})}{\sum_{j=1}^{4} N_c(a_j^2) + N_c(a_1^5)}$$

Replacing N_c by the above values, the costs in Table 7.2 are obtained.

APPENDIX B
Proof of Theorem 5.4.1

To help proving the Theorem, the following proposition will be proven first:

Proposition B.1 *If $C \in \Re$, $C \geq 0$ is a constant, $\alpha(n_i+1) = \frac{1}{n_i+1+C}$ in (3.6), J is given by (4.7) and its estimate by (5.19) or (5.23), Fu's generalized LSA is optimal.*

Proof: For the proposition to be true, the necessary and sufficient condition (3.10) of Theorem 3.1.1 must be satisfied. First, assume that the LSA has m actions, and that (with no loss of generality) the optimal action is u_m. x_i is the state of the environment. Furthermore, the costs will be assumed zero, also with no loss of generality. In fact, a cost different from zero may be interpreted as a bias of the actual reliability. From (5.22) or (5.26)

$$\begin{aligned} E[\lambda_{ij}(n_i)|y(1),\ldots,y(n_{ij})] &= \Pr\{\hat{J}_{ij}(n_{ij}) = \min_q\{\hat{J}_{iq}(n_{iq})\}\} \\ &= \Pr\{\hat{R}_{ij}(n_{ij}) = \max_q\{\hat{R}_{iq}(n_{iq})\}\} \qquad \text{(B.1)} \end{aligned}$$

From Lemma 6.1.1 it can be derived that, for sub-optimal actions u_j

$$\Pr\{\hat{R}_{ij}(n_{ij}) > \hat{R}_{im}(n_{im}),\ j=1,\ldots,m-1\} < 1 - \prod_{j=1}^{m}(1 - \frac{4R_{ij}(1-R_{ij})}{n_{ij}\Delta^2})$$

Noticing that $n_i = \sum_{j=1}^{m} n_{ij}$, one can assume (with no loss of generality) that $n_{ij} = n_i/m$. Denoting terms in negative integer powers of n by $\mathcal{O}\left(\frac{1}{n^k}\right)$, the following equality is obtained:

$$\prod_{j=1}^{m}(1 - \frac{4R_{ij}(1-R_{ij})}{n_{ij}\Delta^2}) = 1 - \frac{4m}{n_i\Delta^2}\sum_{j=1}^{m} R_{ij}(1-R_{ij}) + \sum_{k=2}^{m}\mathcal{O}\left(\frac{1}{n_i^k}\right)$$

Hence,

$$\begin{aligned} \Pr\{\hat{R}_{ij}(n_{ij}) > \hat{R}_{im}(n_{im}),\ \forall j=1,\ldots,m-1\} &< \frac{4m}{n_i\Delta^2}\sum_{j=1}^{m} R_{ij}(1-R_{ij}) + \sum_{k=2}^{m}\mathcal{O}\left(\frac{1}{n_i^k}\right) \\ &\leq \frac{m^2}{n_i\Delta^2} + \sum_{k=2}^{m}\mathcal{O}\left(\frac{1}{n_i^k}\right) \qquad \text{(B.2)} \end{aligned}$$

where the last inequality results from the fact that $0 \leq R_{ij} \leq 1$.

To obtain a reasonable approximation of $\Pr\{\hat{R}_{ij}(n_{ij}) = \max_q\{\hat{R}_{iq}(n_{iq})\}\}$ in Equation (B.1) above, one may admit that the probability of the reliability estimate for a sub-optimal action being greater than the reliability estimate for the optimal action is approximately the same as the probability of the reliability estimate for a sub-optimal action having the maximum value among all reliability estimates:

$$\Pr\{\hat{R}_{ij}(n_{ij}) > \hat{R}_{im}(n_{im}),\ j = 1, \ldots, m-1\} \simeq \Pr\{\hat{R}_{ij}(n_{ij}) = \max_q\{\hat{R}_{iq}(n_{iq}\}\}$$

Given this approximation, Equation (B.1), inequality (B.2), and the expression $\alpha(n_i + 1)$ above,

$$\sum_{n_i=1}^{\infty} \alpha(n_i+1)E[\lambda_{ij}(n_i)|y(1), \ldots, y(n_{ij})] <$$
$$\sum_{n_i=1}^{\infty} \left\{ \frac{m^2}{n_i(n_i+1+C)\Delta^2} + \sum_{k=2}^{m} \mathcal{O}\left(\frac{1}{n_i^k(n_i+1+C)} \right) \right\}$$

The summation on the right of the inequality above converges, hence the condition of Theorem 3.1.1 is satisfied and the Proposition proved. □

By Proposition B.1, every LSA of the **Coordination-to-Execution Translation Level** is optimal, that is when the number of times a task (and consequently the composing primitive events and algorithms) is applied tends to infinity, the probability of selecting the optimal primitive algorithms for each of the primitive events converges to 1 w.p.1. This is valid also for the acceleration scheme described in section 6.2.2, when $C = W$.

To prove the optimality of the **Organization-to-Coordination Translation Level** level LSA, it suffices to show that Proposition B.1 applies to the LSA representing one of the subset of productions $\mathcal{R}_k$ of grammar G for the given command.

First take production $r \in \mathcal{R}_k$, and recall the definitions of production and primitive event reliability

$$\hat{R}_{ir}(n_i) \quad = \quad \prod_{e_k \in \mathcal{E}_r} \hat{R}(e_k, n_i|x_i)$$

$$\hat{R}(e_k, n_i|x_i) \;=\; \sum_{j=1}^{n_k} p_{ij}\hat{R}^k_{ij}(n_{ij}) \;\; \forall e_k \in E$$

Since, by Proposition B.1

$$\Pr\{\lim_{n_i\to\infty} p_{im}(n_i) = 1\} = 1$$

and by the Strong Law of Large Numbers

$$\Pr\{\lim_{n_{ij}\to\infty} \hat{R}^k_{ij}(n_{ij}) = R^k_{ij}\} = 1 \;\; \forall j = 1,\ldots,m$$

there exists some N_i for which

$$\hat{R}(e_k|x_i) \simeq \hat{R}^k_{im}(n_{im}), \quad \forall n_i > N_i \;\; \forall e_k \in E$$

assuming, for simplicity, that all primitive events are translated by an equal number of primitive algorithms m.

Hence,

$$\hat{R}_{ir}(n_i) \simeq \prod_{e_k\in\mathcal{E}_r} \hat{R}^k_{im}(n_{im}), \quad \forall n_i > N_i$$

From the independence of the random variables $\hat{R}^k_{im}(n_{im})$ and recalling that $E[\hat{R}^k_{im}(n_{im})] = R^k_{im}$:

$$\begin{aligned}
\text{var}\;\hat{R}_{ir}(n_i) &= E[\hat{R}^2_{ir}(n_i)] - (E[\hat{R}_{ir}(n_i)])^2 \\
&= E[\prod_{e_k\in\mathcal{E}_r}(\hat{R}^k_{im}(n_{im}))^2] - (E[\prod_{e_k\in\mathcal{E}_r}\hat{R}^k_{im}])^2 \\
&= \prod_{e_k\in\mathcal{E}_r} E[(\hat{R}^k_{im}(n_{im}))^2] - \prod_{e_k\in\mathcal{E}_r}(R^k_{im})^2
\end{aligned}$$

and noticing that

$$E[(\hat{R}^k_{im}(n_{im}))^2] = (R^k_{im})^2 + \frac{R^k_{im}(1 - R^k_{im})}{n_{im}}$$

one finally obtains, making $n_{im} = n_i/m$

$$\text{var}\;\hat{R}_{ir}(n_i) = \sum_{l=1}^{|\mathcal{E}_r|} \mathcal{O}\left(\frac{1}{n_i^l}\right)$$

and an argument similar to that used to prove the convergence of each of the bottom level LSAs applies, using Proposition B.1 and productions as actions of the LSA. Recall that the main argument for the proof was the fact that

$$\text{var}\ \hat{R}_{ij} = \frac{R_{ij}(1 - R_{ij})}{n_{ij}} = \mathcal{O}\left(\frac{1}{n_{ij}}\right), \quad \forall j = 1, \ldots, m$$

Hence, if the reinforcement scheme picks the optimal production for every subset of the set of productions of G, it does actually pick the optimal task for the command, given the current state of the environment, and the Theorem is proved. □